GBATEK

Gameboy Advance / Nintendo DS / DSi - Technical Info - Extracted from no$gba version 3.02

About this Document

GBA Reference

Overview
GBA Technical Data
GBA Memory Map
GBA I/O Map

Hardware Programming
GBA LCD Video Controller
GBA Sound Controller
GBA Timers
GBA DMA Transfers
GBA Communication Ports
GBA Keypad Input
GBA Interrupt Control
GBA System Control
GBA Cartridges
GBA Unpredictable Things

Other
ARM CPU Reference
BIOS Functions
External Connectors

NDS Reference

Overview
DS Technical Data
DS I/O Maps
DS Memory Maps

Hardware Programming
DS Memory Control
DS Video
DS 3D Video
DS Sound
DS System and Built-in Peripherals
DS Cartridges, Encryption, Firmware
DS Xboo
DS Wireless Communications

Other
BIOS Functions
ARM CPU Reference
External Connectors

3DS Reference

3DS Reference (under construction)

DSi Reference

DSi Basic Differences to NDS

New Hardware Features
DSi I/O Map
DSi Control Registers (SCFG)
DSi XpertTeak (DSP)
DSi New Shared WRAM
DSi New DMA (NDMA)
DSi Microphone and SoundExt
DSi Advanced Encryption Standard (AES)
DSi Cartridge Header
DSi Touchscreen/Sound Controller
DSi I2C Bus
DSi Cameras
DSi SD/MMC Protocol and I/O Ports
DSi SD/MMC Filesystem
DSi Atheros Wifi SDIO Interface
DSi Atheros Wifi Internal Hardware
DSi GPIO Registers
DSi Console IDs
DSi Unknown Registers
DSi Notes
DSi Exploits
DSi Regions

CPU Reference

General ARM7TDMI Information
ARM CPU Overview
ARM CPU Register Set
ARM CPU Flags & Condition Field (cond)
ARM CPU 26bit Memory Interface
ARM CPU Exceptions
ARM CPU Memory Alignments

Further Information
ARM Pseudo Instructions and Directives
ARM CP15 System Control

CPU 32bit ARM Mode

ARM 32bit Opcodes (ARM Code)
ARM Instruction Summary
ARM Branch and Branch with Link (B, BL, BX, BLX, SWI, BKPT)
ARM Data Processing (ALU)
ARM Multiply and Multiply-Accumulate (MUL, MLA)
ARM Special ARM9 Instructions (CLZ, QADD/QSUB)
ARM PSR Transfer (MRS, MSR)
ARM Memory: Single Data Transfer (LDR, STR, PLD)
ARM Memory: Halfword, Doubleword, and Signed Data

CPU 16bit THUMB Mode

ARM 16bit Opcodes (THUMB Code)
When operating in THUMB state, cut-down 16bit opcodes are used. THUMB is supported on T-variants of ARMv4 and up, ie. ARMv4T, ARMv5T, etc.
THUMB Instruction Summary
THUMB Register Operations (ALU, BX)
THUMB Memory Load/Store (LDR/STR)
THUMB Memory Addressing

GBA Reference

Overview

Hardware Programming

Other

GBA Technical Data

CPU Modes

ARM Mode	ARM7TDMI 32bit RISC CPU, 16.78MHz, 32bit opcodes (GBA)
THUMB Mode	ARM7TDMI 32bit RISC CPU, 16.78MHz, 16bit opcodes (GBA)
CGB Mode	Z80/8080-style 8bit CPU, 4.2MHz or 8.4MHz (CGB compatibility)
DMG Mode	Z80/8080-style 8bit CPU, 4.2MHz (monochrome gameboy compatib.)

Internal Memory

BIOS ROM	16 KBytes
Work RAM	288 KBytes (Fast 32K on-chip, plus Slow 256K on-board)
VRAM	96 KBytes
OAM	1 KByte (128 OBJs 3x16bit, 32 OBJ-Rotation/Scalings 4x16bit)
Palette RAM	1 KByte (256 BG colors, 256 OBJ colors)

Video

Display	240x160 pixels (2.9 inch TFT color LCD display)
BG layers	4 background layers
BG types	Tile/map based, or Bitmap based
BG colors	256 colors, or 16 colors/16 palettes, or 32768 colors
OBJ colors	256 colors, or 16 colors/16 palettes
OBJ size	12 types (in range 8x8 up to 64x64 dots)
OBJs/Screen	max. 128 OBJs of any size (up to 64x64 dots each)
OBJs/Line	max. 128 OBJs of 8x8 dots size (under best circumstances)
Priorities	OBJ/OBJ: 0-127, OBJ/BG: 0-3, BG/BG: 0-3

Effects	Rotation/Scaling, alpha blending, fade-in/out, mosaic, window
Backlight	GBA SP only (optionally by light on/off toggle button)

Sound

Analogue	4 channel CGB compatible (3x square wave, 1x noise)
Digital	2 DMA sound channels
Output	Built-in speaker (mono), or headphones socket (stereo)

Controls

Gamepad	4 Direction Keys, 6 Buttons

Communication Ports

Serial Port	Various transfer modes, 4-Player Link, Single Game Pak play

External Memory

GBA Game Pak max. 32MB ROM or flash ROM + max 64K SRAM
CGB Game Pak max. 32KB ROM + 8KB SRAM (more memory requires banking)

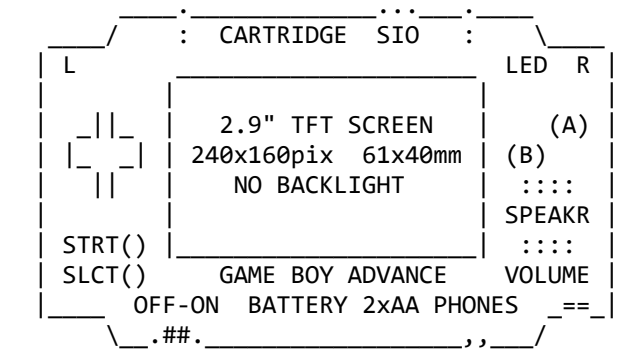
Case Dimensions

Size (mm)	GBA: 145x81x25 - GBA SP: 82x82x24 (closed), 155x82x24 (stretch)

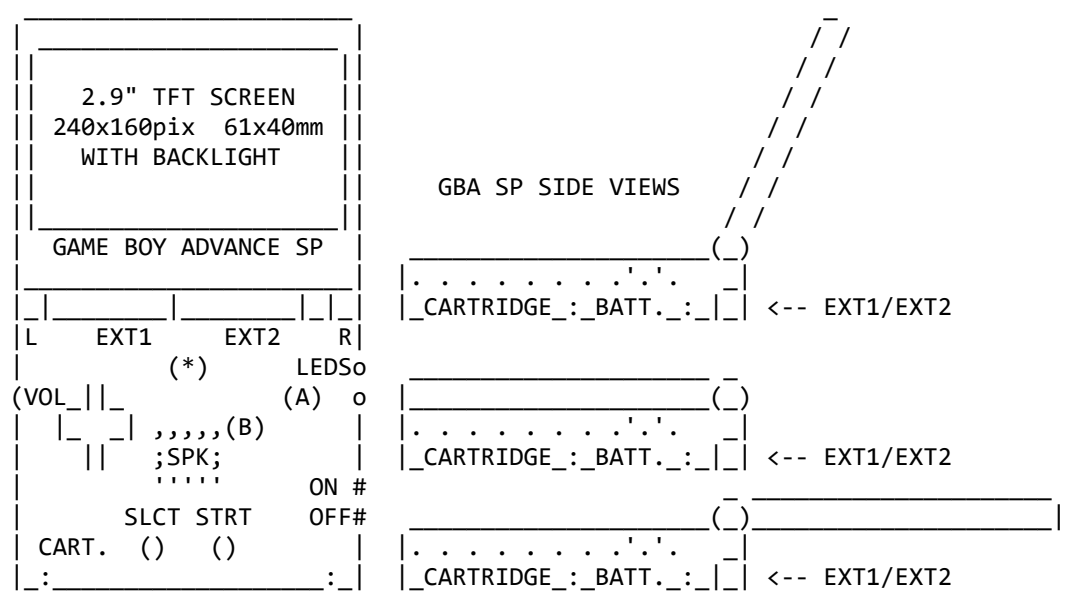
Power Supply

Battery GBA	GBA: 2x1.5V DC (AA), Life-time approx. 15 hours
Battery SP	GBA SP: Built-in rechargeable Lithium ion battery, 3.7V 600mAh
External	GBA: 3.3V DC 350mA - GBA SP: 5.2V DC 320mA

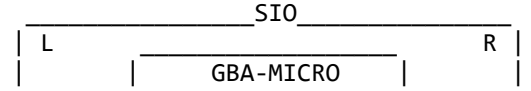
Original Gameboy Advance (GBA)



GBA SP (GBA SP)



Gameboy Micro (GBA Micro)



```
    | _||_  |  2.0" TFT SCREEN |    (A)| +
    ||_  _| |240x160pix 42x28mm| (B)   |VOL
    |  ||   |     BACKLIGHT    |       | -
    |       |__________________|  ...  |
    |___________SELECT__START___________|
      PWR   <--- CARTRIDGE SLOT ---> PHONES
```

Nintendo DS (NDS)

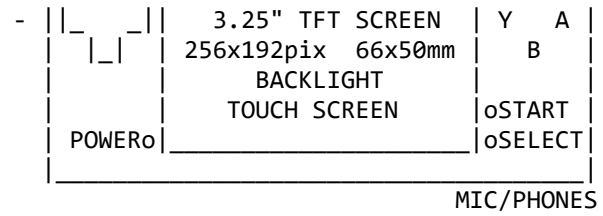
```
     ____________________________________
    |                                    |
    |        ____________________        |
    |       |                    |       |
    |       |    3" TFT SCREEN   |       |
    |       | 256x192pix  61x46mm|       |
    |       |      BACKLIGHT     |       |
    | ::::: |    Original NDS    | ::::: |
    | ::::: |____________________| ::::: |
  _|         _           ______   _       |_  <-- gap between screens: 22mm
 |L|_______| |________|      |_| |_______|R|     (equivalent to 90 pixels)
 |_______                                 |
 |  PWR  | |                    | |SEL STA|
 |   _   | |    3" TFT SCREEN   | |       |
 | _| |_ | | 256x192pix  61x46mm | |   X   |
 ||_   _|| |      BACKLIGHT     | | Y   A |
 |  |_|  | |    TOUCH SCREEN    | |   B   |
 |       | |____________________| |       |
 |_______|          NintendoDS    |_______|
 |        MIC              LEDS           |
 |____________________________________|
     VOL       SLOT2(GBA)     MIC/PHONES
```

Nintendo DS Lite (NDS-Lite)

```
     ____________________________________
    |                                    |
    |        ____________________        |
    |       |                    |       |
    |       |    3" TFT SCREEN   |       |
    |  ...  | 256x192pix  61x46mm |  ...  |
    |  ...  |      BACKLIGHT     |  ...  |
    |       |      NDS-LITE      |       |
    |       |____________________|       |
    |___  _ _ _ _ _ _ _ _ _ _ _ _ ____|   <-- gap between screens: 23mm
  L|  _ |_____________MIC_____________|LEDS|R
    |   _    ____________________        |
    | _| |_ |                    |   X   |
    ||_   _||    3" TFT SCREEN   | Y   A |PWR
    |  |_|  | 256x192pix  61x46mm |   B   |
    |       |      BACKLIGHT     |       |
    |       |    TOUCH SCREEN    |oSTART |
    |       |____________________|oSELECT|
    |____________________________________|
     VOL       SLOT2(GBA)     MIC/PHONES
```

Nintendo DSi (DSi)

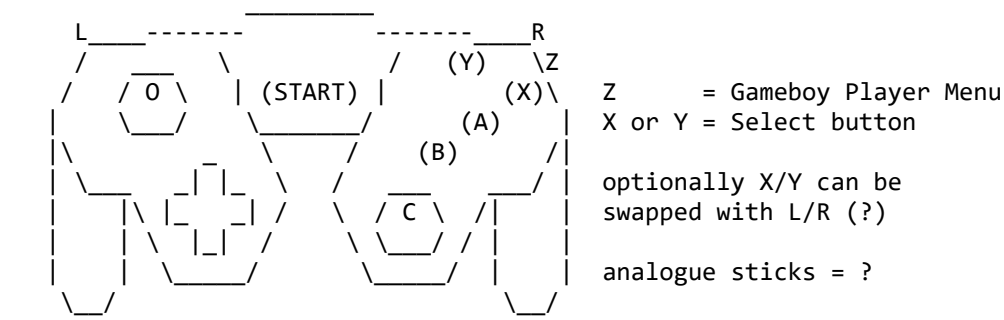
```
     ____________________________________
    |                                    |
    |        ____________________        |
    |       |                    |   O o | <-- CAM (O) and LED (o)
    |       |   3.25" TFT SCREEN |       |     (on backside)
    |       | 256x192pix  66x50mm |       |
    |       |      BACKLIGHT     |       |
    |  __   |         DSi        |  __   |
    | (__)  |____________________| (__)  |
    |___  _ _ _ _ _ _ _ _ _ _ _ _ ____|  <-- gap between screens: 23mm
  L|LEDS|__________CAM__MIC_________| __ |R                   (88 pixels)
 +  |   _    ____________________        |
VOL| _| |_ |                    |   X   | <-- SD Card Slot
```



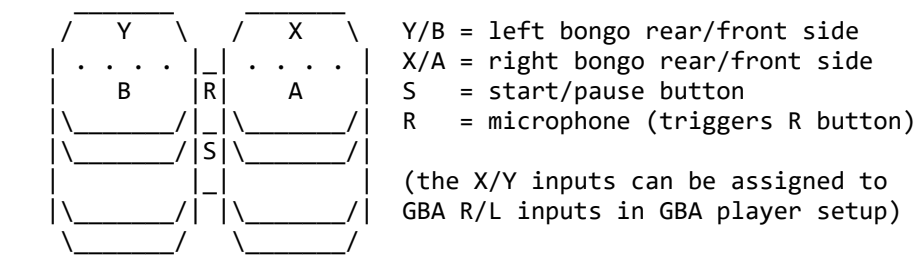
Nintendo DSi XL

As DSi, but bigger case, and bigger 4.2" screens

Gameboy Player (Gamecube Joypad) (GBA Player)



Gameboy Player (Gamecube Bongos) (GBA Player)



The GBA's separate 8bit/32bit CPU modes cannot be operated simultaneously. Switching is allowed between ARM and THUMB modes only (that are the two GBA modes).
This manual does not describe CGB and DMG modes, both are completely different than GBA modes, and both cannot be accessed from inside of GBA modes anyways.

Gameboy Player

An GBA Adapter for the Gamecube console; allowing to play GBA games on a television set.
GBA Gameboy Player

GBA SP Notes

Deluxe version of the original GBA. With backlight, new folded laptop-style case, and built-in rechargeable battery. Appears to be 100% compatible with GBA, there seems to be no way to detect SPs by software.

Gameboy Micro (GBA Micro)

Minituarized GBA. Supports 32bit GBA games only (no 8bit DMG/CGB games). The 256K Main RAM is a bit slower than usually (cannot be "overclocked via port 4000800h).

Nintendo DS (Dual Screen) Notes

New handheld with two screens, backwards compatible with GBA games, it is NOT backwards compatible with older 8bit games (mono/color gameboys) though..
Also, the DS has no link port, so that GBA games will thus work only in single player mode, link-port accessoires like printers cannot be used, and most unfortunately multiboot won't work (trying to press Select+Start at powerup will just lock up the DS).

iQue Notes
iQue is a brand name used by Nintendo in China, iQue GBA and iQue DS are essentially same as Nintendo GBA and Nintendo DS.
The iQue DS contains a larger firmware chip (the charset additionally contains about 6700 simplified chinese characters), the bootmenu still allows to select (only) six languages (japanese has been replaced by chinese). The iQue DS can play normal international NDS games, plus chinese dedicated games. The latter ones won't work on normal NDS consoles (that, reportedly simply due to a firmware-version check contained in chinese dedicated games, aside from that check, the games should be fully compatible with NDS consoles).

GBA Memory Map

General Internal Memory
```
00000000-00003FFF   BIOS - System ROM         (16 KBytes)
00004000-01FFFFFF   Not used
02000000-0203FFFF   WRAM - On-board Work RAM  (256 KBytes) 2 Wait
02040000-02FFFFFF   Not used
03000000-03007FFF   WRAM - On-chip Work RAM   (32 KBytes)
03008000-03FFFFFF   Not used
04000000-040003FE   I/O Registers
04000400-04FFFFFF   Not used
```
Internal Display Memory
```
05000000-050003FF   BG/OBJ Palette RAM        (1 Kbyte)
05000400-05FFFFFF   Not used
06000000-06017FFF   VRAM - Video RAM          (96 KBytes)
06018000-06FFFFFF   Not used
07000000-070003FF   OAM - OBJ Attributes      (1 Kbyte)
07000400-07FFFFFF   Not used
```
External Memory (Game Pak)
```
08000000-09FFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 0
0A000000-0BFFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 1
0C000000-0DFFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 2
0E000000-0E00FFFF   Game Pak SRAM    (max 64 KBytes) - 8bit Bus width
0E010000-0FFFFFFF   Not used
```
Unused Memory Area
```
10000000-FFFFFFFF   Not used (upper 4bits of address bus unused)
```

Default WRAM Usage
By default, the 256 bytes at 03007F00h-03007FFFh in Work RAM are reserved for Interrupt vector, Interrupt Stack, and BIOS Call Stack. The remaining WRAM is free for whatever use (including User Stack, which is initially located at 03007F00h).

Address Bus Width and CPU Read/Write Access Widths
Shows the Bus-Width, supported read and write widths, and the clock cycles for 8/16/32bit accesses.
```
Region        Bus   Read      Write     Cycles
BIOS ROM      32    8/16/32   -         1/1/1
Work RAM 32K  32    8/16/32   8/16/32   1/1/1
I/O           32    8/16/32   8/16/32   1/1/1
OAM           32    8/16/32   16/32     1/1/1 *
Work RAM 256K 16    8/16/32   8/16/32   3/3/6 **
Palette RAM   16    8/16/32   16/32     1/1/2 *
VRAM          16    8/16/32   16/32     1/1/2 *
GamePak ROM   16    8/16/32   -         5/5/8 **/***
GamePak Flash 16    8/16/32   16/32     5/5/8 **/***
GamePak SRAM  8     8         8         5     **
```
Timing Notes:
```
*   Plus 1 cycle if GBA accesses video memory at the same time.
**  Default waitstate settings, see System Control chapter.
```

```
  *** Separate timings for sequential, and non-sequential accesses.
  One cycle equals approx. 59.59ns (ie. 16.78MHz clock).
```
All memory (except GamePak SRAM) can be accessed by 16bit and 32bit DMA.

GamePak Memory
Only DMA3 (and the CPU of course) may access GamePak ROM. GamePak SRAM can be accessed by the CPU only - restricted to bytewise 8bit transfers. The SRAM region is supposed for as external FLASH backup memory, or for battery-backed SRAM.
For details about configuration of GamePak Waitstates, see:
GBA System Control

VRAM, OAM, and Palette RAM Access
These memory regions can be accessed during H-Blank or V-Blank only (unless display is disabled by Forced Blank bit in DISPCNT register).
There is an additional restriction for OAM memory: Accesses during H-Blank are allowed only if 'H-Blank Interval Free' in DISPCNT is set (which'd reduce number of display-able OBJs though).
The CPU appears to be able to access VRAM/OAM/Palette at any time, a waitstate (one clock cycle) being inserted automatically in case that the display controller was accessing memory simultaneously. (Ie. unlike as in old 8bit gameboy, the data will not get lost.)

CPU Mode Performance
Note that the GamePak ROM bus is limited to 16bits, thus executing ARM instructions (32bit opcodes) from inside of GamePak ROM would result in a not so good performance. So, it'd be more recommended to use THUMB instruction (16bit opcodes) which'd allow each opcode to be read at once.
(ARM instructions can be used at best performance by copying code from GamePak ROM into internal Work RAM)

Data Format
Even though the ARM CPU itself would allow to select between Little-Endian and Big-Endian format by using an external circuit, in the GBA no such circuit exists, and the data format is always Little-Endian. That is, when accessing 16bit or 32bit data in memory, the least significant bits are stored in the first byte (smallest address), and the most significant bits in the last byte. (Ie. same as for 80x86 and Z80 CPUs.)

GBA I/O Map

LCD I/O Registers

```
  4000000h  2    R/W  DISPCNT   LCD Control
  4000002h  2    R/W  -         Undocumented - Green Swap
  4000004h  2    R/W  DISPSTAT  General LCD Status (STAT,LYC)
  4000006h  2    R    VCOUNT    Vertical Counter (LY)
  4000008h  2    R/W  BG0CNT    BG0 Control
  400000Ah  2    R/W  BG1CNT    BG1 Control
  400000Ch  2    R/W  BG2CNT    BG2 Control
  400000Eh  2    R/W  BG3CNT    BG3 Control
  4000010h  2    W    BG0HOFS   BG0 X-Offset
  4000012h  2    W    BG0VOFS   BG0 Y-Offset
  4000014h  2    W    BG1HOFS   BG1 X-Offset
  4000016h  2    W    BG1VOFS   BG1 Y-Offset
  4000018h  2    W    BG2HOFS   BG2 X-Offset
  400001Ah  2    W    BG2VOFS   BG2 Y-Offset
  400001Ch  2    W    BG3HOFS   BG3 X-Offset
  400001Eh  2    W    BG3VOFS   BG3 Y-Offset
  4000020h  2    W    BG2PA     BG2 Rotation/Scaling Parameter A (dx)
  4000022h  2    W    BG2PB     BG2 Rotation/Scaling Parameter B (dmx)
  4000024h  2    W    BG2PC     BG2 Rotation/Scaling Parameter C (dy)
  4000026h  2    W    BG2PD     BG2 Rotation/Scaling Parameter D (dmy)
```

```
4000028h  4    W    BG2X      BG2 Reference Point X-Coordinate
400002Ch  4    W    BG2Y      BG2 Reference Point Y-Coordinate
4000030h  2    W    BG3PA     BG3 Rotation/Scaling Parameter A (dx)
4000032h  2    W    BG3PB     BG3 Rotation/Scaling Parameter B (dmx)
4000034h  2    W    BG3PC     BG3 Rotation/Scaling Parameter C (dy)
4000036h  2    W    BG3PD     BG3 Rotation/Scaling Parameter D (dmy)
4000038h  4    W    BG3X      BG3 Reference Point X-Coordinate
400003Ch  4    W    BG3Y      BG3 Reference Point Y-Coordinate
4000040h  2    W    WIN0H     Window 0 Horizontal Dimensions
4000042h  2    W    WIN1H     Window 1 Horizontal Dimensions
4000044h  2    W    WIN0V     Window 0 Vertical Dimensions
4000046h  2    W    WIN1V     Window 1 Vertical Dimensions
4000048h  2    R/W  WININ     Inside of Window 0 and 1
400004Ah  2    R/W  WINOUT    Inside of OBJ Window & Outside of Windows
400004Ch  2    W    MOSAIC    Mosaic Size
400004Eh       -    -         Not used
4000050h  2    R/W  BLDCNT    Color Special Effects Selection
4000052h  2    R/W  BLDALPHA  Alpha Blending Coefficients
4000054h  2    W    BLDY      Brightness (Fade-In/Out) Coefficient
4000056h       -    -         Not used
```

Sound Registers

```
4000060h  2  R/W  SOUND1CNT_L Channel 1 Sweep register       (NR10)
4000062h  2  R/W  SOUND1CNT_H Channel 1 Duty/Length/Envelope (NR11, NR12)
4000064h  2  R/W  SOUND1CNT_X Channel 1 Frequency/Control    (NR13, NR14)
4000066h     -    -           Not used
4000068h  2  R/W  SOUND2CNT_L Channel 2 Duty/Length/Envelope (NR21, NR22)
400006Ah     -    -           Not used
400006Ch  2  R/W  SOUND2CNT_H Channel 2 Frequency/Control    (NR23, NR24)
400006Eh     -    -           Not used
4000070h  2  R/W  SOUND3CNT_L Channel 3 Stop/Wave RAM select (NR30)
4000072h  2  R/W  SOUND3CNT_H Channel 3 Length/Volume        (NR31, NR32)
4000074h  2  R/W  SOUND3CNT_X Channel 3 Frequency/Control    (NR33, NR34)
4000076h     -    -           Not used
4000078h  2  R/W  SOUND4CNT_L Channel 4 Length/Envelope      (NR41, NR42)
400007Ah     -    -           Not used
400007Ch  2  R/W  SOUND4CNT_H Channel 4 Frequency/Control    (NR43, NR44)
400007Eh     -    -           Not used
4000080h  2  R/W  SOUNDCNT_L  Control Stereo/Volume/Enable   (NR50, NR51)
4000082h  2  R/W  SOUNDCNT_H  Control Mixing/DMA Control
4000084h  2  R/W  SOUNDCNT_X  Control Sound on/off           (NR52)
4000086h     -    -           Not used
4000088h  2  BIOS SOUNDBIAS   Sound PWM Control
400008Ah  ..   -    -         Not used
4000090h 2x10h R/W  WAVE_RAM  Channel 3 Wave Pattern RAM (2 banks!!)
40000A0h  4    W    FIFO_A    Channel A FIFO, Data 0-3
40000A4h  4    W    FIFO_B    Channel B FIFO, Data 0-3
40000A8h       -    -         Not used
```

DMA Transfer Channels

```
40000B0h  4    W    DMA0SAD   DMA 0 Source Address
40000B4h  4    W    DMA0DAD   DMA 0 Destination Address
40000B8h  2    W    DMA0CNT_L DMA 0 Word Count
40000BAh  2    R/W  DMA0CNT_H DMA 0 Control
40000BCh  4    W    DMA1SAD   DMA 1 Source Address
40000C0h  4    W    DMA1DAD   DMA 1 Destination Address
40000C4h  2    W    DMA1CNT_L DMA 1 Word Count
40000C6h  2    R/W  DMA1CNT_H DMA 1 Control
40000C8h  4    W    DMA2SAD   DMA 2 Source Address
40000CCh  4    W    DMA2DAD   DMA 2 Destination Address
40000D0h  2    W    DMA2CNT_L DMA 2 Word Count
40000D2h  2    R/W  DMA2CNT_H DMA 2 Control
40000D4h  4    W    DMA3SAD   DMA 3 Source Address
40000D8h  4    W    DMA3DAD   DMA 3 Destination Address
40000DCh  2    W    DMA3CNT_L DMA 3 Word Count
```

```
  40000DEh  2    R/W  DMA3CNT_H DMA 3 Control
  40000E0h       -    -         Not used
```
Timer Registers
```
  4000100h  2    R/W  TM0CNT_L  Timer 0 Counter/Reload
  4000102h  2    R/W  TM0CNT_H  Timer 0 Control
  4000104h  2    R/W  TM1CNT_L  Timer 1 Counter/Reload
  4000106h  2    R/W  TM1CNT_H  Timer 1 Control
  4000108h  2    R/W  TM2CNT_L  Timer 2 Counter/Reload
  400010Ah  2    R/W  TM2CNT_H  Timer 2 Control
  400010Ch  2    R/W  TM3CNT_L  Timer 3 Counter/Reload
  400010Eh  2    R/W  TM3CNT_H  Timer 3 Control
  4000110h       -    -         Not used
```
Serial Communication (1)
```
  4000120h  4    R/W  SIODATA32 SIO Data (Normal-32bit Mode; shared with below)
  4000120h  2    R/W  SIOMULTI0 SIO Data 0 (Parent)    (Multi-Player Mode)
  4000122h  2    R/W  SIOMULTI1 SIO Data 1 (1st Child) (Multi-Player Mode)
  4000124h  2    R/W  SIOMULTI2 SIO Data 2 (2nd Child) (Multi-Player Mode)
  4000126h  2    R/W  SIOMULTI3 SIO Data 3 (3rd Child) (Multi-Player Mode)
  4000128h  2    R/W  SIOCNT    SIO Control Register
  400012Ah  2    R/W  SIOMLT_SEND SIO Data (Local of MultiPlayer; shared below)
  400012Ah  2    R/W  SIODATA8  SIO Data (Normal-8bit and UART Mode)
  400012Ch       -    -         Not used
```
Keypad Input
```
  4000130h  2    R    KEYINPUT  Key Status
  4000132h  2    R/W  KEYCNT    Key Interrupt Control
```
Serial Communication (2)
```
  4000134h  2    R/W  RCNT      SIO Mode Select/General Purpose Data
  4000136h  -    -    IR        Ancient - Infrared Register (Prototypes only)
  4000138h       -    -         Not used
  4000140h  2    R/W  JOYCNT    SIO JOY Bus Control
  4000142h       -    -         Not used
  4000150h  4    R/W  JOY_RECV  SIO JOY Bus Receive Data
  4000154h  4    R/W  JOY_TRANS SIO JOY Bus Transmit Data
  4000158h  2    R/?  JOYSTAT   SIO JOY Bus Receive Status
  400015Ah       -    -         Not used
```
Interrupt, Waitstate, and Power-Down Control
```
  4000200h  2    R/W  IE        Interrupt Enable Register
  4000202h  2    R/W  IF        Interrupt Request Flags / IRQ Acknowledge
  4000204h  2    R/W  WAITCNT   Game Pak Waitstate Control
  4000206h       -    -         Not used
  4000208h  2    R/W  IME       Interrupt Master Enable Register
  400020Ah       -    -         Not used
  4000300h  1    R/W  POSTFLG   Undocumented - Post Boot Flag
  4000301h  1    W    HALTCNT   Undocumented - Power Down Control
  4000302h       -    -         Not used
  4000410h  ?    ?    ?         Undocumented - Purpose Unknown / Bug ??? 0FFh
  4000411h       -    -         Not used
  4000800h  4    R/W  ?         Undocumented - Internal Memory Control (R/W)
  4000804h       -    -         Not used
  4xx0800h  4    R/W  ?         Mirrors of 4000800h (repeated each 64K)
  4700000h  4    W    (3DS)     Disable ARM7 bootrom overlay (3DS only)
```

All further addresses at 4XXXXXXh are unused and do not contain mirrors of the I/O area, with the only exception that 4000800h is repeated each 64K (ie. mirrored at 4010800h, 4020800h, etc.)

GBA LCD Video Controller

Registers
LCD I/O Display Control
LCD I/O Interrupts and Status

VRAM

Sprites

Other

LCD I/O Display Control

4000000h - DISPCNT - LCD Control (Read/Write)

```
Bit   Expl.
0-2   BG Mode                (0-5=Video Mode 0-5, 6-7=Prohibited)
3     Reserved / CGB Mode    (0=GBA, 1=CGB; can be set only by BIOS opcodes)
4     Display Frame Select   (0-1=Frame 0-1) (for BG Modes 4,5 only)
5     H-Blank Interval Free  (1=Allow access to OAM during H-Blank)
6     OBJ Character VRAM Mapping (0=Two dimensional, 1=One dimensional)
7     Forced Blank           (1=Allow FAST access to VRAM,Palette,OAM)
8     Screen Display BG0  (0=Off, 1=On)
9     Screen Display BG1  (0=Off, 1=On)
10    Screen Display BG2  (0=Off, 1=On)
11    Screen Display BG3  (0=Off, 1=On)
12    Screen Display OBJ  (0=Off, 1=On)
13    Window 0 Display Flag   (0=Off, 1=On)
14    Window 1 Display Flag   (0=Off, 1=On)
15    OBJ Window Display Flag (0=Off, 1=On)
```

The table summarizes the facilities of the separate BG modes (video modes).

Mode	Rot/Scal	Layers	Size	Tiles	Colors	Features
0	No	0123	256x256..512x515	1024	16/16..256/1	SFMABP
1	Mixed	012-	(BG0,BG1 as above Mode 0, BG2 as below Mode 2)			
2	Yes	--23	128x128..1024x1024	256	256/1	S-MABP
3	Yes	--2-	240x160	1	32768	--MABP
4	Yes	--2-	240x160	2	256/1	--MABP
5	Yes	--2-	160x128	2	32768	--MABP

Features: S)crolling, F)lip, M)osaic, A)lphaBlending, B)rightness, P)riority.

BG Modes 0-2 are Tile/Map-based. BG Modes 3-5 are Bitmap-based, in these modes 1 or 2 Frames (ie. bitmaps, or 'full screen tiles') exists, if two frames exist, either one can be displayed, and the other one can be redrawn in background.

Blanking Bits
Setting Forced Blank (Bit 7) causes the video controller to display white lines, and all VRAM, Palette RAM, and OAM may be accessed.
"When the internal HV synchronous counter cancels a forced blank during a display period, the display begins from the beginning, following the display of two vertical lines." What ?
Setting H-Blank Interval Free (Bit 5) allows to access OAM during H-Blank time - using this feature reduces the number of sprites that can be displayed per line.

Display Enable Bits
By default, BG0-3 and OBJ Display Flags (Bit 8-12) are used to enable/disable BGs and OBJ. When enabling Window 0 and/or 1 (Bit 13-14), color special effects may be used, and BG0-3 and OBJ are controlled by the window(s).

Frame Selection
In BG Modes 4 and 5 (Bitmap modes), either one of the two bitmaps/frames may be displayed (Bit 4), allowing the user to update the other (invisible) frame in background. In BG Mode 3, only one frame exists.
In BG Modes 0-2 (Tile/Map based modes), a similar effect may be gained by altering the base address(es) of BG Map and/or BG Character data.

4000002h - Undocumented - Green Swap (R/W)
Normally, red green blue intensities for a group of two pixels is output as BGRbgr (uppercase for left pixel at even xloc, lowercase for right pixel at odd xloc). When the Green Swap bit is set, each pixel group is output as BgRbGr (ie. green intensity of each two pixels exchanged).

Bit	Expl.
0	Green Swap (0=Normal, 1=Swap)
1-15	Not used

This feature appears to be applied to the final picture (ie. after mixing the separate BG and OBJ layers). Eventually intended for other display types (with other pin-outs). With normal GBA hardware it is just producing an interesting dirt effect.
The NDS DISPCNT registers are 32bit (4000000h..4000003h), so Green Swap doesn't exist in NDS mode, however, the NDS does support Green Swap in GBA mode.

LCD I/O Interrupts and Status

4000004h - DISPSTAT - General LCD Status (Read/Write)
Display status and Interrupt control. The H-Blank conditions are generated once per scanline, including for the 'hidden' scanlines during V-Blank.

Bit	Expl.		
0	V-Blank flag	(Read only) (1=VBlank) (set in line 160..226; not 227)	
1	H-Blank flag	(Read only) (1=HBlank) (toggled in all lines, 0..227)	
2	V-Counter flag	(Read only) (1=Match) (set in selected line)	(R)
3	V-Blank IRQ Enable	(1=Enable)	(R/W)
4	H-Blank IRQ Enable	(1=Enable)	(R/W)
5	V-Counter IRQ Enable	(1=Enable)	(R/W)
6	Not used (0) / DSi: LCD Initialization Ready (0=Busy, 1=Ready)		(R)
7	Not used (0) / NDS: MSB of V-Vcount Setting (LYC.Bit8) (0..262)		(R/W)
8-15	V-Count Setting (LYC)	(0..227)	(R/W)

The V-Count-Setting value is much the same as LYC of older gameboys, when its value is identical to the content of the VCOUNT register then the V-Counter flag is set (Bit 2), and (if enabled in Bit 5) an interrupt is requested.
Although the drawing time is only 960 cycles (240*4), the H-Blank flag is "0" for a total of 1006 cycles.

4000006h - VCOUNT - Vertical Counter (Read only)
Indicates the currently drawn scanline, values in range from 160..227 indicate 'hidden' scanlines within VBlank area.

```
  Bit   Expl.
  0-7   Current Scanline (LY)      (0..227)                              (R)
  8     Not used (0) / NDS: MSB of Current Scanline (LY.Bit8) (0..262)   (R)
  9-15  Not Used (0)
```
Note: This is much the same than the 'LY' register of older gameboys.

LCD I/O BG Control

4000008h - BG0CNT - BG0 Control (R/W) (BG Modes 0,1 only)
400000Ah - BG1CNT - BG1 Control (R/W) (BG Modes 0,1 only)
400000Ch - BG2CNT - BG2 Control (R/W) (BG Modes 0,1,2 only)
400000Eh - BG3CNT - BG3 Control (R/W) (BG Modes 0,2 only)
```
  Bit   Expl.
  0-1   BG Priority           (0-3, 0=Highest)
  2-3   Character Base Block  (0-3, in units of 16 KBytes) (=BG Tile Data)
  4-5   Not used (must be zero) (except in NDS mode: MSBs of char base)
  6     Mosaic                (0=Disable, 1=Enable)
  7     Colors/Palettes       (0=16/16, 1=256/1)
  8-12  Screen Base Block     (0-31, in units of 2 KBytes) (=BG Map Data)
  13    BG0/BG1: Not used (except in NDS mode: Ext Palette Slot for BG0/BG1)
  13    BG2/BG3: Display Area Overflow (0=Transparent, 1=Wraparound)
  14-15 Screen Size (0-3)
```
Internal Screen Size (dots) and size of BG Map (bytes):
```
  Value  Text Mode      Rotation/Scaling Mode
  0      256x256 (2K)   128x128   (256 bytes)
  1      512x256 (4K)   256x256   (1K)
  2      256x512 (4K)   512x512   (4K)
  3      512x512 (8K)   1024x1024 (16K)
```
In case that some or all BGs are set to same priority then BG0 is having the highest, and BG3 the lowest priority.

In 'Text Modes', the screen size is organized as follows: The screen consists of one or more 256x256 pixel (32x32 tiles) areas. When Size=0: only 1 area (SC0), when Size=1 or Size=2: two areas (SC0,SC1 either horizontally or vertically arranged next to each other), when Size=3: four areas (SC0,SC1 in upper row, SC2,SC3 in lower row). Whereas SC0 is defined by the normal BG Map base address (Bit 8-12 of BGxCNT), SC1 uses same address +2K, SC2 address +4K, SC3 address +6K. When the screen is scrolled it'll always wraparound.

In 'Rotation/Scaling Modes', the screen size is organized as follows, only one area (SC0) of variable size 128x128..1024x1024 pixels (16x16..128x128 tiles) exists. When the screen is rotated/scaled (or scrolled?) so that the LCD viewport reaches outside of the background/screen area, then BG may be either displayed as transparent or wraparound (Bit 13 of BGxCNT).

LCD I/O BG Scrolling

4000010h - BG0HOFS - BG0 X-Offset (W)
4000012h - BG0VOFS - BG0 Y-Offset (W)
```
  Bit   Expl.
  0-8   Offset (0-511)
  9-15  Not used
```
Specifies the coordinate of the upperleft first visible dot of BG0 background layer, ie. used to scroll the BG0 area.

4000014h - BG1HOFS - BG1 X-Offset (W)
4000016h - BG1VOFS - BG1 Y-Offset (W)
Same as above BG0HOFS and BG0VOFS for BG1 respectively.

4000018h - BG2HOFS - BG2 X-Offset (W)
400001Ah - BG2VOFS - BG2 Y-Offset (W)
Same as above BG0HOFS and BG0VOFS for BG2 respectively.

400001Ch - BG3HOFS - BG3 X-Offset (W)
400001Eh - BG3VOFS - BG3 Y-Offset (W)
Same as above BG0HOFS and BG0VOFS for BG3 respectively.

The above BG scrolling registers are exclusively used in Text modes, ie. for all layers in BG Mode 0, and for the first two layers in BG mode 1.
In other BG modes (Rotation/Scaling and Bitmap modes) above registers are ignored. Instead, the screen may be scrolled by modifying the BG Rotation/Scaling Reference Point registers.

LCD I/O BG Rotation/Scaling

4000028h - BG2X_L - BG2 Reference Point X-Coordinate, lower 16 bit (W)
400002Ah - BG2X_H - BG2 Reference Point X-Coordinate, upper 12 bit (W)
400002Ch - BG2Y_L - BG2 Reference Point Y-Coordinate, lower 16 bit (W)
400002Eh - BG2Y_H - BG2 Reference Point Y-Coordinate, upper 12 bit (W)
These registers are replacing the BG scrolling registers which are used for Text mode, ie. the X/Y coordinates specify the source position from inside of the BG Map/Bitmap of the pixel to be displayed at upper left of the GBA display. The normal BG scrolling registers are ignored in Rotation/Scaling and Bitmap modes.

```
Bit   Expl.
0-7   Fractional portion (8 bits)
8-26  Integer portion    (19 bits)
27    Sign               (1 bit)
28-31 Not used
```

Because values are shifted left by eight, fractional portions may be specified in steps of 1/256 pixels (this would be relevant only if the screen is actually rotated or scaled). Normal signed 32bit values may be written to above registers (the most significant bits will be ignored and the value will be cut-down to 28bits, but this is no actual problem because signed values have set all MSBs to the same value).

Internal Reference Point Registers
The above reference points are automatically copied to internal registers during each vblank, specifying the origin for the first scanline. The internal registers are then incremented by dmx and dmy after each scanline. Caution: Writing to a reference point register by software outside of the Vblank period does immediately copy the new value to the corresponding internal register, that means: in the current frame, the new value specifies the origin of the <current> scanline (instead of the topmost scanline).

4000020h - BG2PA - BG2 Rotation/Scaling Parameter A (alias dx) (W)
4000022h - BG2PB - BG2 Rotation/Scaling Parameter B (alias dmx) (W)
4000024h - BG2PC - BG2 Rotation/Scaling Parameter C (alias dy) (W)
4000026h - BG2PD - BG2 Rotation/Scaling Parameter D (alias dmy) (W)

```
Bit   Expl.
0-7   Fractional portion (8 bits)
8-14  Integer portion    (7 bits)
15    Sign               (1 bit)
```

See below for details.

400003Xh - BG3X_L/H, BG3Y_L/H, BG3PA-D - BG3 Rotation/Scaling Parameters
Same as above BG2 Reference Point, and Rotation/Scaling Parameters, for BG3 respectively.

dx (PA) and dy (PC)

When transforming a horizontal line, dx and dy specify the resulting gradient and magnification for that line. For example:
Horizontal line, length=100, dx=1, and dy=1. The resulting line would be drawn at 45 degrees, f(y)=1/1*x. Note that this would involve that line is magnified, the new length is SQR(100^2+100^2)=141.42. Yup, exactly - that's the old a^2 + b^2 = c^2 formula.

dmx (PB) and dmy (PD)
These values define the resulting gradient and magnification for transformation of vertical lines. However, when rotating a square area (which is surrounded by horizontal and vertical lines), then the desired result should be usually a rotated <square> area (ie. not a parallelogram, for example).
Thus, dmx and dmy must be defined in direct relationship to dx and dy, taking the example above, we'd have to set dmx=-1, and dmy=1, f(x)=-1/1*y.

Area Overflow
In result of rotation/scaling it may often happen that areas outside of the actual BG area become moved into the LCD viewport. Depending of the Area Overflow bit (BG2CNT and BG3CNT, Bit 13) these areas may be either displayed (by wrapping the BG area), or may be displayed transparent.
This works only in BG modes 1 and 2. The area overflow is ignored in Bitmap modes (BG modes 3-5), the outside of the Bitmaps is always transparent.

--- more details and confusing or helpful formulas ---

The following parameters are required for Rotation/Scaling
```
Rotation Center X and Y Coordinates (x0,y0)
Rotation Angle                      (alpha)
Magnification X and Y Values        (xMag,yMag)
```
The display is rotated by 'alpha' degrees around the center.
The displayed picture is magnified by 'xMag' along x-Axis (Y=y0) and 'yMag' along y-Axis (X=x0).

Calculating Rotation/Scaling Parameters A-D
```
A = Cos (alpha) / xMag    ;distance moved in direction x, same line
B = Sin (alpha) / xMag    ;distance moved in direction x, next line
C = Sin (alpha) / yMag    ;distance moved in direction y, same line
D = Cos (alpha) / yMag    ;distance moved in direction y, next line
```

Calculating the position of a rotated/scaled dot
Using the following expressions,
```
x0,y0    Rotation Center
x1,y1    Old Position of a pixel (before rotation/scaling)
x2,y2    New position of above pixel (after rotation scaling)
A,B,C,D  BG2PA-BG2PD Parameters (as calculated above)
```
the following formula can be used to calculate x2,y2:
```
x2 = A(x1-x0) + B(y1-y0) + x0
y2 = C(x1-x0) + D(y1-y0) + y0
```

LCD I/O Window Feature

The Window Feature may be used to split the screen into four regions. The BG0-3,OBJ layers and Color Special Effects can be separately enabled or disabled in each of these regions.

The DISPCNT Register
DISPCNT Bits 13-15 are used to enable Window 0, Window 1, and/or OBJ Window regions, if any of these regions is enabled then the "Outside of Windows" region is automatically enabled, too.
DISPCNT Bits 8-12 are kept used as master enable bits for the BG0-3,OBJ layers, a layer is displayed only if both DISPCNT and WININ/OUT enable bits are set.

4000040h - WIN0H - Window 0 Horizontal Dimensions (W)
4000042h - WIN1H - Window 1 Horizontal Dimensions (W)

```
Bit   Expl.
0-7   X2, Rightmost coordinate of window, plus 1
8-15  X1, Leftmost coordinate of window
```

Garbage values of X2>240 or X1>X2 are interpreted as X2=240.

4000044h - WIN0V - Window 0 Vertical Dimensions (W)
4000046h - WIN1V - Window 1 Vertical Dimensions (W)

```
Bit   Expl.
0-7   Y2, Bottom-most coordinate of window, plus 1
8-15  Y1, Top-most coordinate of window
```

Garbage values of Y2>160 or Y1>Y2 are interpreted as Y2=160.

4000048h - WININ - Control of Inside of Window(s) (R/W)

```
Bit   Expl.
0-3   Window 0 BG0-BG3 Enable Bits     (0=No Display, 1=Display)
4     Window 0 OBJ Enable Bit          (0=No Display, 1=Display)
5     Window 0 Color Special Effect    (0=Disable, 1=Enable)
6-7   Not used
8-11  Window 1 BG0-BG3 Enable Bits     (0=No Display, 1=Display)
12    Window 1 OBJ Enable Bit          (0=No Display, 1=Display)
13    Window 1 Color Special Effect    (0=Disable, 1=Enable)
14-15 Not used
```

400004Ah - WINOUT - Control of Outside of Windows & Inside of OBJ Window (R/W)

```
Bit   Expl.
0-3   Outside BG0-BG3 Enable Bits      (0=No Display, 1=Display)
4     Outside OBJ Enable Bit           (0=No Display, 1=Display)
5     Outside Color Special Effect     (0=Disable, 1=Enable)
6-7   Not used
8-11  OBJ Window BG0-BG3 Enable Bits   (0=No Display, 1=Display)
12    OBJ Window OBJ Enable Bit        (0=No Display, 1=Display)
13    OBJ Window Color Special Effect  (0=Disable, 1=Enable)
14-15 Not used
```

The OBJ Window
The dimension of the OBJ Window is specified by OBJs which are having the "OBJ Mode" attribute being set to "OBJ Window". Any non-transparent dots of any such OBJs are marked as OBJ Window area. The OBJ itself is not displayed.
The color, palette, and display priority of these OBJs are ignored. Both DISPCNT Bits 12 and 15 must be set when defining OBJ Window region(s).

Window Priority
In case that more than one window is enabled, and that these windows do overlap, Window 0 is having highest priority, Window 1 medium, and Obj Window lowest priority. Outside of Window is having zero priority, it is used for all dots which are not inside of any window region.

LCD I/O Mosaic Function

400004Ch - MOSAIC - Mosaic Size (W)
The Mosaic function can be separately enabled/disabled for BG0-BG3 by BG0CNT-BG3CNT Registers, as well as for each OBJ0-127 by OBJ attributes in OAM memory. Also, setting all of the bits below to zero effectively disables the mosaic function.

```
Bit   Expl.
0-3   BG Mosaic H-Size  (minus 1)
```

```
  4-7   BG Mosaic V-Size  (minus 1)
  8-11  OBJ Mosaic H-Size (minus 1)
  12-15 OBJ Mosaic V-Size (minus 1)
  16-31 Not used
```
Example: When setting H-Size to 5, then pixels 0-5 of each display row are colorized as pixel 0, pixels 6-11 as pixel 6, pixels 12-17 as pixel 12, and so on.

Normally, a 'mosaic-pixel' is colorized by the color of the upperleft covered pixel. In many cases it might be more desireful to use the color of the pixel in the center of the covered area - this effect may be gained by scrolling the background (or by adjusting the OBJ position, as far as upper/left rows/columns of OBJ are transparent).

LCD I/O Color Special Effects

Two types of Special Effects are supported: Alpha Blending (Semi-Transparency) allows to combine colors of two selected surfaces. Brightness Increase/Decrease adjust the brightness of the selected surface.

4000050h - BLDCNT - Color Special Effects Selection (R/W)
```
  Bit   Expl.
  0     BG0 1st Target Pixel (Background 0)
  1     BG1 1st Target Pixel (Background 1)
  2     BG2 1st Target Pixel (Background 2)
  3     BG3 1st Target Pixel (Background 3)
  4     OBJ 1st Target Pixel (Top-most OBJ pixel)
  5     BD  1st Target Pixel (Backdrop)
  6-7   Color Special Effect (0-3, see below)
         0 = None                (Special effects disabled)
         1 = Alpha Blending      (1st+2nd Target mixed)
         2 = Brightness Increase (1st Target becomes whiter)
         3 = Brightness Decrease (1st Target becomes blacker)
  8     BG0 2nd Target Pixel (Background 0)
  9     BG1 2nd Target Pixel (Background 1)
  10    BG2 2nd Target Pixel (Background 2)
  11    BG3 2nd Target Pixel (Background 3)
  12    OBJ 2nd Target Pixel (Top-most OBJ pixel)
  13    BD  2nd Target Pixel (Backdrop)
  14-15 Not used
```
Selects the 1st Target layer(s) for special effects. For Alpha Blending/Semi-Transparency, it does also select the 2nd Target layer(s), which should have next lower display priority as the 1st Target.
However, any combinations are possible, including that all layers may be selected as both 1st+2nd target, in that case the top-most pixel will be used as 1st target, and the next lower pixel as 2nd target.

4000052h - BLDALPHA - Alpha Blending Coefficients (R/W) (not W)
Used for Color Special Effects Mode 1, and for Semi-Transparent OBJs.
```
  Bit   Expl.
  0-4   EVA Coefficient (1st Target) (0..16 = 0/16..16/16, 17..31=16/16)
  5-7   Not used
  8-12  EVB Coefficient (2nd Target) (0..16 = 0/16..16/16, 17..31=16/16)
  13-15 Not used
```
For this effect, the top-most non-transparent pixel must be selected as 1st Target, and the next-lower non-transparent pixel must be selected as 2nd Target, if so - and only if so, then color intensities of 1st and 2nd Target are mixed together by using the parameters in BLDALPHA register, for each pixel each R, G, B intensities are calculated separately:
```
  I = MIN ( 31, I1st*EVA + I2nd*EVB )
```
Otherwise - for example, if only one target exists, or if a non-transparent non-2nd-target pixel is moved between the two targets, or if 2nd target has higher display priority than 1st target - then only the top-most pixel is displayed (at normal intensity, regardless of BLDALPHA).

4000054h - BLDY - Brightness (Fade-In/Out) Coefficient (W) (not R/W)
Used for Color Special Effects Modes 2 and 3.

```
Bit   Expl.
0-4   EVY Coefficient (Brightness) (0..16 = 0/16..16/16, 17..31=16/16)
5-31  Not used
```

For each pixel each R, G, B intensities are calculated separately:

```
I = I1st + (31-I1st)*EVY   ;For Brightness Increase
I = I1st - (I1st)*EVY      ;For Brightness Decrease
```

The color intensities of any selected 1st target surface(s) are increased or decreased by using the parameter in BLDY register.

Semi-Transparent OBJs
OBJs that are defined as 'Semi-Transparent' in OAM memory are always selected as 1st Target (regardless of BLDCNT Bit 4), and are always using Alpha Blending mode (regardless of BLDCNT Bit 6-7).
The BLDCNT register may be used to perform Brightness effects on the OBJ (and/or other BG/BD layers). However, if a semi-transparent OBJ pixel does overlap a 2nd target pixel, then semi-transparency becomes priority, and the brightness effect will not take place (neither on 1st, nor 2nd target).

The OBJ Layer
Before special effects are applied, the display controller computes the OBJ priority ordering, and isolates the top-most OBJ pixel. In result, only the top-most OBJ pixel is recursed at the time when processing special effects. Ie. alpha blending and semi-transparency can be used for OBJ-to-BG or BG-to-OBJ , but not for OBJ-to-OBJ.

LCD VRAM Overview

The GBA contains 96 Kbytes VRAM built-in, located at address 06000000-06017FFF, depending on the BG Mode used as follows:

BG Mode 0,1,2 (Tile/Map based Modes)

```
06000000-0600FFFF  64 KBytes shared for BG Map and Tiles
06010000-06017FFF  32 KBytes OBJ Tiles
```

The shared 64K area can be split into BG Map area(s), and BG Tiles area(s), the respective addresses for Map and Tile areas are set up by BG0CNT-BG3CNT registers. The Map address may be specified in units of 2K (steps of 800h), the Tile address in units of 16K (steps of 4000h).

BG Mode 0,1 (Tile/Map based Text mode)
The tiles may have 4bit or 8bit color depth, minimum map size is 32x32 tiles, maximum is 64x64 tiles, up to 1024 tiles can be used per map.

```
Item        Depth     Required Memory
One Tile    4bit      20h bytes
One Tile    8bit      40h bytes
1024 Tiles  4bit      8000h (32K)
1024 Tiles  8bit      10000h (64K) - excluding some bytes for BG map
BG Map      32x32     800h (2K)
BG Map      64x64     2000h (8K)
```

BG Mode 1,2 (Tile/Map based Rotation/Scaling mode)
The tiles may have 8bit color depth only, minimum map size is 16x16 tiles, maximum is 128x128 tiles, up to 256 tiles can be used per map.

```
Item        Depth     Required Memory
One Tile    8bit      40h bytes
256  Tiles  8bit      4000h (16K)
BG Map      16x16     100h bytes
BG Map      128x128   4000h (16K)
```

BG Mode 3 (Bitmap based Mode for still images)

```
06000000-06013FFF  80 KBytes Frame 0 buffer (only 75K actually used)
06014000-06017FFF  16 KBytes OBJ Tiles
```

BG Mode 4,5 (Bitmap based Modes)

```
06000000-06009FFF  40 KBytes Frame 0 buffer (only 37.5K used in Mode 4)
0600A000-06013FFF  40 KBytes Frame 1 buffer (only 37.5K used in Mode 4)
06014000-06017FFF  16 KBytes OBJ Tiles
```

Note

Additionally to the above VRAM, the GBA also contains 1 KByte Palette RAM (at 05000000h) and 1 KByte OAM (at 07000000h) which are both used by the display controller as well.

LCD VRAM Character Data

Each character (tile) consists of 8x8 dots (64 dots in total). The color depth may be either 4bit or 8bit (see BG0CNT-BG3CNT).

4bit depth (16 colors, 16 palettes)

Each tile occupies 32 bytes of memory, the first 4 bytes for the topmost row of the tile, and so on. Each byte representing two dots, the lower 4 bits define the color for the left (!) dot, the upper 4 bits the color for the right dot.

8bit depth (256 colors, 1 palette)

Each tile occupies 64 bytes of memory, the first 8 bytes for the topmost row of the tile, and so on. Each byte selects the palette entry for each dot.

LCD VRAM BG Screen Data Format (BG Map)

The display background consists of 8x8 dot tiles, the arrangement of these tiles is specified by the BG Screen Data (BG Map). The separate entries in this map are as follows:

Text BG Screen (2 bytes per entry)

Specifies the tile number and attributes. Note that BG tile numbers are always specified in steps of 1 (unlike OBJ tile numbers which are using steps of two in 256 color/1 palette mode).

```
Bit   Expl.
0-9   Tile Number     (0-1023) (a bit less in 256 color mode, because
                         there'd be otherwise no room for the bg map)
10    Horizontal Flip (0=Normal, 1=Mirrored)
11    Vertical Flip   (0=Normal, 1=Mirrored)
12-15 Palette Number  (0-15)    (Not used in 256 color/1 palette mode)
```

A Text BG Map always consists of 32x32 entries (256x256 pixels), 400h entries = 800h bytes. However, depending on the BG Size, one, two, or four of these Maps may be used together, allowing to create backgrounds of 256x256, 512x256, 256x512, or 512x512 pixels, if so, the first map (SC0) is located at base+0, the next map (SC1) at base+800h, and so on.

Rotation/Scaling BG Screen (1 byte per entry)

In this mode, only 256 tiles can be used. There are no x/y-flip attributes, the color depth is always 256 colors/1 palette.

```
Bit   Expl.
0-7   Tile Number     (0-255)
```

The dimensions of Rotation/Scaling BG Maps depend on the BG size. For size 0-3 that are: 16x16 tiles (128x128 pixels), 32x32 tiles (256x256 pixels), 64x64 tiles (512x512 pixels), or 128x128 tiles (1024x1024 pixels).

The size and VRAM base address of the separate BG maps for BG0-3 are set up by BG0CNT-BG3CNT registers.

LCD VRAM Bitmap BG Modes

In BG Modes 3-5 the background is defined in form of a bitmap (unlike as for Tile/Map based BG modes). Bitmaps are implemented as BG2, with Rotation/Scaling support. As bitmap modes are occupying 80KBytes of BG memory, only 16KBytes of VRAM can be used for OBJ tiles.

BG Mode 3 - 240x160 pixels, 32768 colors
Two bytes are associated to each pixel, directly defining one of the 32768 colors (without using palette data, and thus not supporting a 'transparent' BG color).

```
Bit   Expl.
0-4   Red Intensity   (0-31)
5-9   Green Intensity (0-31)
10-14 Blue Intensity  (0-31)
15    Not used in GBA Mode (in NDS Mode: Alpha=0=Transparent, Alpha=1=Normal)
```

The first 480 bytes define the topmost line, the next 480 the next line, and so on. The background occupies 75 KBytes (06000000-06012BFF), most of the 80 Kbytes BG area, not allowing to redraw an invisible second frame in background, so this mode is mostly recommended for still images only.

BG Mode 4 - 240x160 pixels, 256 colors (out of 32768 colors)
One byte is associated to each pixel, selecting one of the 256 palette entries. Color 0 (backdrop) is transparent, and OBJs may be displayed behind the bitmap.
The first 240 bytes define the topmost line, the next 240 the next line, and so on. The background occupies 37.5 KBytes, allowing two frames to be used (06000000-060095FF for Frame 0, and 0600A000-060135FF for Frame 1).

BG Mode 5 - 160x128 pixels, 32768 colors
Colors are defined as for Mode 3 (see above), but horizontal and vertical size are cut down to 160x128 pixels only - smaller than the physical dimensions of the LCD screen.
The background occupies exactly 40 KBytes, so that BG VRAM may be split into two frames (06000000-06009FFF for Frame 0, and 0600A000-06013FFF for Frame 1).

In BG modes 4,5, one Frame may be displayed (selected by DISPCNT Bit 4), the other Frame is invisible and may be redrawn in background.

LCD OBJ - Overview

General
Objects (OBJs) are moveable sprites. Up to 128 OBJs (of any size, up to 64x64 dots each) can be displayed per screen, and under best circumstances up to 128 OBJs (of small 8x8 dots size) can be displayed per horizontal display line.

Maximum Number of Sprites per Line
The total available OBJ rendering cycles per line are

```
1210  (=304*4-6)   If "H-Blank Interval Free" bit in DISPCNT register is 0
954   (=240*4-6)   If "H-Blank Interval Free" bit in DISPCNT register is 1
```

The required rendering cycles are (depending on horizontal OBJ size)

```
Cycles per <n> Pixels    OBJ Type              OBJ Type Screen Pixel Range
n*1 cycles               Normal OBJs           8..64 pixels
10+n*2 cycles            Rotation/Scaling OBJs 8..64 pixels   (area clipped)
10+n*2 cycles            Rotation/Scaling OBJs 16..128 pixels (double size)
```

Caution:
The maximum number of OBJs per line is also affected by undisplayed (offscreen) OBJs which are having higher priority than displayed OBJs.
To avoid this, move displayed OBJs to the begin of OAM memory (ie. OBJ0 has highest priority, OBJ127 lowest).
Otherwise (in case that the program logic expects OBJs at fixed positions in OAM) at least take care to set the OBJ size of undisplayed OBJs to 8x8 with Rotation/Scaling disabled (this reduces the overload).
Does the above also apply for VERTICALLY OFFSCREEN (or VERTICALLY not on CURRENT LINE) sprites ?

VRAM - Character Data
OBJs are always combined of one or more 8x8 pixel Tiles (much like BG Tiles in BG Modes 0-2). However, OBJ Tiles are stored in a separate area in VRAM: 06010000-06017FFF (32 KBytes) in BG Mode 0-2, or 06014000-06017FFF (16 KBytes) in BG Mode 3-5.
Depending on the size of the above area (16K or 32K), and on the OBJ color depth (4bit or 8bit), 256-1024 8x8 dots OBJ Tiles can be defined.

OAM - Object Attribute Memory
This memory area contains Attributes which specify position, size, color depth, etc. appearance for each of the 128 OBJs. Additionally, it contains 32 OBJ Rotation/Scaling Parameter groups. OAM is located at 07000000-070003FF (sized 1 KByte).

LCD OBJ - OAM Attributes

OBJ Attributes
There are 128 entries in OAM for each OBJ0-OBJ127. Each entry consists of 6 bytes (three 16bit Attributes). Attributes for OBJ0 are located at 07000000, for OBJ1 at 07000008, OBJ2 at 07000010, and so on.

As you can see, there are blank spaces at 07000006, 0700000E, 07000016, etc. - these 16bit values are used for OBJ Rotation/Scaling (as described in the next chapter) - they are not directly related to the separate OBJs.

OBJ Attribute 0 (R/W)

```
Bit   Expl.
0-7   Y-Coordinate           (0-255)
8     Rotation/Scaling Flag  (0=Off, 1=On)
When Rotation/Scaling used (Attribute 0, bit 8 set):
  9     Double-Size Flag     (0=Normal, 1=Double)
When Rotation/Scaling not used (Attribute 0, bit 8 cleared):
  9     OBJ Disable          (0=Normal, 1=Not displayed)
10-11 OBJ Mode  (0=Normal, 1=Semi-Transparent, 2=OBJ Window, 3=Prohibited)
12    OBJ Mosaic             (0=Off, 1=On)
13    Colors/Palettes        (0=16/16, 1=256/1)
14-15 OBJ Shape              (0=Square,1=Horizontal,2=Vertical,3=Prohibited)
```

Caution: A very large OBJ (of 128 pixels vertically, ie. a 64 pixels OBJ in a Double Size area) located at Y>128 will be treated as at Y>-128, the OBJ is then displayed parts offscreen at the TOP of the display, it is then NOT displayed at the bottom.

OBJ Attribute 1 (R/W)

```
Bit   Expl.
0-8   X-Coordinate           (0-511)
```

```
When Rotation/Scaling used (Attribute 0, bit 8 set):
  9-13  Rotation/Scaling Parameter Selection (0-31)
        (Selects one of the 32 Rotation/Scaling Parameters that
        can be defined in OAM, for details read next chapter.)
When Rotation/Scaling not used (Attribute 0, bit 8 cleared):
  9-11  Not used
  12    Horizontal Flip      (0=Normal, 1=Mirrored)
  13    Vertical Flip        (0=Normal, 1=Mirrored)
14-15 OBJ Size               (0..3, depends on OBJ Shape, see Attr 0)
        Size  Square   Horizontal  Vertical
        0     8x8      16x8        8x16
        1     16x16    32x8        8x32
        2     32x32    32x16       16x32
        3     64x64    64x32       32x64
```

OBJ Attribute 2 (R/W)

```
Bit   Expl.
0-9   Character Name          (0-1023=Tile Number)
10-11 Priority relative to BG (0-3; 0=Highest)
12-15 Palette Number   (0-15) (Not used in 256 color/1 palette mode)
```

Notes:

OBJ Mode
The OBJ Mode may be Normal, Semi-Transparent, or OBJ Window.
Semi-Transparent means that the OBJ is used as 'Alpha Blending 1st Target' (regardless of BLDCNT register, for details see chapter about Color Special Effects).
OBJ Window means that the OBJ is not displayed, instead, dots with non-zero color are used as mask for the OBJ Window, see DISPCNT and WINOUT for details.

OBJ Tile Number
There are two situations which may divide the amount of available tiles by two (by four if both situations apply):

1. When using the 256 Colors/1 Palette mode, only each second tile may be used, the lower bit of the tile number should be zero (in 2-dimensional mapping mode, the bit is completely ignored).

2. When using BG Mode 3-5 (Bitmap Modes), only tile numbers 512-1023 may be used. That is because lower 16K of OBJ memory are used for BG. Attempts to use tiles 0-511 are ignored (not displayed).

Priority
In case that the 'Priority relative to BG' is the same than the priority of one of the background layers, then the OBJ becomes higher priority and is displayed on top of that BG layer.
Caution: Take care not to mess up BG Priority and OBJ priority. For example, the following would cause garbage to be displayed:

```
OBJ No. 0 with Priority relative to BG=1   ;hi OBJ prio, lo BG prio
OBJ No. 1 with Priority relative to BG=0   ;lo OBJ prio, hi BG prio
```

That is, OBJ0 is always having priority above OBJ1-127, so assigning a lower BG Priority to OBJ0 than for OBJ1-127 would be a bad idea.

LCD OBJ - OAM Rotation/Scaling Parameters

As described in the previous chapter, there are blank spaces between each of the 128 OBJ Attribute Fields in OAM memory. These 128 16bit gaps are used to store OBJ Rotation/Scaling Parameters.

Location of Rotation/Scaling Parameters in OAM

Four 16bit parameters (PA,PB,PC,PD) are required to define a complete group of Rotation/Scaling data. These are spread across OAM as such:

```
1st Group - PA=07000006, PB=0700000E, PC=07000016, PD=0700001E
2nd Group - PA=07000026, PB=0700002E, PC=07000036, PD=0700003E
etc.
```

By using all blank space (128 x 16bit), up to 32 of these groups (4 x 16bit each) can be defined in OAM.

OBJ Rotation/Scaling PA,PB,PC,PD Parameters (R/W)
Each OBJ that uses Rotation/Scaling may select between any of the above 32 parameter groups. For details, refer to the previous chapter about OBJ Attributes.
The meaning of the separate PA,PB,PC,PD values is identical as for BG, for details read the chapter about BG Rotation/Scaling.

OBJ Reference Point & Rotation Center
The OBJ Reference Point is the upper left of the OBJ, ie. OBJ X/Y coordinates: X+0, Y+0.
The OBJ Rotation Center is always (or should be usually?) in the middle of the object, ie. for a 8x32 pixel OBJ, this would be at the OBJ X/Y coordinates: X+4, and Y+16.

OBJ Double-Size Bit (for OBJs that use Rotation/Scaling)
When Double-Size is zero: The sprite is rotated, and then display inside of the normal-sized (not rotated) rectangular area - the edges of the rotated sprite will become invisible if they reach outside of that area.
When Double-Size is set: The sprite is rotated, and then display inside of the double-sized (not rotated) rectangular area - this ensures that the edges of the rotated sprite remain visible even if they would reach outside of the normal-sized area. (Except that, for example, rotating a 8x32 pixel sprite by 90 degrees would still cut off parts of the sprite as the double-size area isn't large enough.)

LCD OBJ - VRAM Character (Tile) Mapping

Each OBJ tile consists of 8x8 dots, however, bigger OBJs can be displayed by combining several 8x8 tiles. The horizontal and vertical size for each OBJ may be separately defined in OAM, possible H/V sizes are 8,16,32,64 dots - allowing 'square' OBJs to be used (such like 8x8, 16x16, etc) as well as 'rectangular' OBJs (such like 8x32, 64x16, etc.)

When displaying an OBJ that contains of more than one 8x8 tile, one of the following two mapping modes can be used. In either case, the tile number of the upperleft tile must be specified in OAM memory.

Two Dimensional Character Mapping (DISPCNT Bit 6 cleared)
This mapping mode assumes that the 1024 OBJ tiles are arranged as a matrix of 32x32 tiles / 256x256 pixels (In 256 color mode: 16x32 tiles / 128x256 pixels). Ie. the upper row of this matrix contains tiles 00h-1Fh, the next row tiles 20h-3Fh, and so on.
For example, when displaying a 16x16 pixel OBJ, with tile number set to 04h; The upper row of the OBJ will consist of tile 04h and 05h, the next row of 24h and 25h. (In 256 color mode: 04h and 06h, 24h and 26h.)

One Dimensional Character Mapping (DISPCNT Bit 6 set)
In this mode, tiles are mapped each after each other from 00h-3FFh.
Using the same example as above, the upper row of the OBJ will consist of tile 04h and 05h, the next row of tile 06h and 07h. (In 256 color mode: 04h and 06h, 08h and 0Ah.)

LCD Color Palettes

Color Palette RAM

BG and OBJ palettes are using separate memory regions:

```
05000000-050001FF - BG Palette RAM (512 bytes, 256 colors)
05000200-050003FF - OBJ Palette RAM (512 bytes, 256 colors)
```

Each BG and OBJ palette RAM may be either split into 16 palettes with 16 colors each, or may be used as a single palette with 256 colors.
Note that some OBJs may access palette RAM in 16 color mode, while other OBJs may use 256 color mode at the same time. Same for BG0-BG3 layers.

Transparent Colors
Color 0 of all BG and OBJ palettes is transparent. Even though palettes are described as 16 (256) color palettes, only 15 (255) colors are actually visible.

Backdrop Color
Color 0 of BG Palette 0 is used as backdrop color. This color is displayed if an area of the screen is not covered by any non-transparent BG or OBJ dots.

Color Definitions
Each color occupies two bytes (same as for 32768 color BG modes):

```
Bit   Expl.
0-4   Red Intensity   (0-31)
5-9   Green Intensity (0-31)
10-14 Blue Intensity  (0-31)
15    Not used
```

Intensities
Under normal circumstances (light source/viewing angle), the intensities 0-14 are practically all black, and only intensities 15-31 are resulting in visible medium..bright colors.

Note: The intensity problem appears in the 8bit CGB "compatibility" mode either. The original CGB display produced the opposite effect: Intensities 0-14 resulted in dark..medium colors, and intensities 15-31 resulted in bright colors. Any "medium" colors of CGB games will appear invisible/black on GBA hardware, and only very bright colors will be visible.

LCD Dimensions and Timings

Horizontal Dimensions
The drawing time for each dot is 4 CPU cycles.

```
Visible     240 dots,  57.221 us,    960 cycles - 78% of h-time
H-Blanking   68 dots,  16.212 us,    272 cycles - 22% of h-time
Total       308 dots,  73.433 us,   1232 cycles - ca. 13.620 kHz
```

VRAM and Palette RAM may be accessed during H-Blanking. OAM can accessed only if "H-Blank Interval Free" bit in DISPCNT register is set.

Vertical Dimensions

```
Visible (*) 160 lines, 11.749 ms, 197120 cycles - 70% of v-time
V-Blanking   68 lines,  4.994 ms,  83776 cycles - 30% of v-time
Total       228 lines, 16.743 ms, 280896 cycles - ca. 59.737 Hz
```

All VRAM, OAM, and Palette RAM may be accessed during V-Blanking.
Note that no H-Blank interrupts are generated within V-Blank period.

System Clock
The system clock is 16.78MHz (16*1024*1024 Hz), one cycle is thus approx. 59.59ns.

(*) Even though vertical screen size is 160 lines, the upper 8 lines are not <really> visible, these lines are covered by a shadow when holding the GBA orientated towards a light source, the lines are effectively black -

and should not be used to display important information.

Interlace
The LCD display is using some sort of interlace in which even scanlines are dimmed in each second frame, and odd scanlines are dimmed in each other frame (it does always render ALL lines in ALL frames, but half of them are dimmed).
The effect can be seen when displaying some horizontal lines in each second frame, and hiding them in each other frame: the hardware will randomly show the lines in dimmed or non-dimmed form (depending on whether the test was started in an even or odd frame).
Unknown if it's possible to determine the even/off frame state by software (or possibly to reset the hardware to this or that state by software).
Note: The NDS is applying some sort of frameskip to GBA games, about every 3 seconds there will by a missing (or maybe: inserted) frame, ie. a GBA game that is updating the display in sync with GBA interlace will get offsync on NDS consoles.

GBA Sound Controller

The GBA supplies four 'analogue' sound channels for Tone and Noise (mostly compatible to CGB sound), as well as two 'digital' sound channels (which can be used to replay 8bit DMA sample data).

GBA Sound Channel 1 - Tone & Sweep
GBA Sound Channel 2 - Tone
GBA Sound Channel 3 - Wave Output
GBA Sound Channel 4 - Noise
GBA Sound Channel A and B - DMA Sound

GBA Sound Control Registers
GBA Comparison of CGB and GBA Sound

The GBA includes only a single (mono) speaker built-in, each channel may be output to either left and/or right channels by using the external line-out connector (for stereo headphones, etc).

GBA Sound Channel 1 - Tone & Sweep

4000060h - SOUND1CNT_L (NR10) - Channel 1 Sweep register (R/W)

Bit		Expl.	
0-2	R/W	Number of sweep shift	(n=0-7)
3	R/W	Sweep Frequency Direction	(0=Increase, 1=Decrease)
4-6	R/W	Sweep Time; units of 7.8ms	(0-7, min=7.8ms, max=54.7ms)
7-15	-	Not used	

Sweep is disabled by setting Sweep Time to zero, if so, the direction bit should be set.
The change of frequency (NR13,NR14) at each shift is calculated by the following formula where X(0) is initial freq & X(t-1) is last freq:

```
X(t) = X(t-1) +/- X(t-1)/2^n
```

4000062h - SOUND1CNT_H (NR11, NR12) - Channel 1 Duty/Len/Envelope (R/W)

Bit		Expl.	
0-5	W	Sound length; units of (64-n)/256s	(0-63)
6-7	R/W	Wave Pattern Duty	(0-3, see below)
8-10	R/W	Envelope Step-Time; units of n/64s	(1-7, 0=No Envelope)
11	R/W	Envelope Direction	(0=Decrease, 1=Increase)
12-15	R/W	Initial Volume of envelope	(1-15, 0=No Sound)

Wave Duty:

```
  0: 12.5% ( -_______-_______-_______ )
  1: 25%   ( --______--______--______ )
  2: 50%   ( ----____----____----____ ) (normal)
  3: 75%   ( ------__------__------__ )
```
The Length value is used only if Bit 6 in NR14 is set.

4000064h - SOUND1CNT_X (NR13, NR14) - Channel 1 Frequency/Control (R/W)
```
  Bit        Expl.
  0-10  W    Frequency; 131072/(2048-n)Hz  (0-2047)
  11-13 -    Not used
  14    R/W  Length Flag  (1=Stop output when length in NR11 expires)
  15    W    Initial      (1=Restart Sound)
  16-31 -    Not used
```

GBA Sound Channel 2 - Tone

This sound channel works exactly as channel 1, except that it doesn't have a Tone Envelope/Sweep Register.

4000068h - SOUND2CNT_L (NR21, NR22) - Channel 2 Duty/Length/Envelope (R/W)
400006Ah - Not used
400006Ch - SOUND2CNT_H (NR23, NR24) - Channel 2 Frequency/Control (R/W)
For details, refer to channel 1 description.

GBA Sound Channel 3 - Wave Output

This channel can be used to output digital sound, the length of the sample buffer (Wave RAM) can be either 32 or 64 digits (4bit samples). This sound channel can be also used to output normal tones when initializing the Wave RAM by a square wave. This channel doesn't have a volume envelope register.

4000070h - SOUND3CNT_L (NR30) - Channel 3 Stop/Wave RAM select (R/W)
```
  Bit        Expl.
  0-4   -    Not used
  5     R/W  Wave RAM Dimension   (0=One bank/32 digits, 1=Two banks/64 digits)
  6     R/W  Wave RAM Bank Number (0-1, see below)
  7     R/W  Sound Channel 3 Off  (0=Stop, 1=Playback)
  8-15  -    Not used
```
The currently selected Bank Number (Bit 6) will be played back, while reading/writing to/from wave RAM will address the other (not selected) bank. When dimension is set to two banks, output will start by replaying the currently selected bank.

4000072h - SOUND3CNT_H (NR31, NR32) - Channel 3 Length/Volume (R/W)
```
  Bit        Expl.
  0-7   W    Sound length; units of (256-n)/256s  (0-255)
  8-12  -    Not used.
  13-14 R/W  Sound Volume  (0=Mute/Zero, 1=100%, 2=50%, 3=25%)
  15    R/W  Force Volume  (0=Use above, 1=Force 75% regardless of above)
```
The Length value is used only if Bit 6 in NR34 is set.

4000074h - SOUND3CNT_X (NR33, NR34) - Channel 3 Frequency/Control (R/W)
```
  Bit        Expl.
  0-10  W    Sample Rate; 2097152/(2048-n) Hz   (0-2047)
  11-13 -    Not used
  14    R/W  Length Flag  (1=Stop output when length in NR31 expires)
  15    W    Initial      (1=Restart Sound)
  16-31 -    Not used
```

The above sample rate specifies the number of wave RAM digits per second, the actual tone frequency depends on the wave RAM content, for example:

```
Wave RAM, single bank 32 digits   Tone Frequency
FFFFFFFFFFFFFFFF0000000000000000  65536/(2048-n) Hz
FFFFFFFF00000000FFFFFFFF00000000  131072/(2048-n) Hz
FFFF0000FFFF0000FFFF0000FFFF0000  262144/(2048-n) Hz
FF00FF00FF00FF00FF00FF00FF00FF00  524288/(2048-n) Hz
F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0  1048576/(2048-n) Hz
```

4000090h - WAVE_RAM0_L - Channel 3 Wave Pattern RAM (W/R)
4000092h - WAVE_RAM0_H - Channel 3 Wave Pattern RAM (W/R)
4000094h - WAVE_RAM1_L - Channel 3 Wave Pattern RAM (W/R)
4000096h - WAVE_RAM1_H - Channel 3 Wave Pattern RAM (W/R)
4000098h - WAVE_RAM2_L - Channel 3 Wave Pattern RAM (W/R)
400009Ah - WAVE_RAM2_H - Channel 3 Wave Pattern RAM (W/R)
400009Ch - WAVE_RAM3_L - Channel 3 Wave Pattern RAM (W/R)
400009Eh - WAVE_RAM3_H - Channel 3 Wave Pattern RAM (W/R)
This area contains 16 bytes (32 x 4bits) Wave Pattern data which is output by channel 3. Data is played back ordered as follows: MSBs of 1st byte, followed by LSBs of 1st byte, followed by MSBs of 2nd byte, and so on - this results in a confusing ordering when filling Wave RAM in units of 16bit data - ie. samples would be then located in Bits 4-7, 0-3, 12-15, 8-11.

In the GBA, two Wave Patterns exists (each 32 x 4bits), either one may be played (as selected in NR30 register), the other bank may be accessed by the users. After all 32 samples have been played, output of the same bank (or other bank, as specified in NR30) will be automatically restarted.

Internally, Wave RAM is a giant shift-register, there is no pointer which is addressing the currently played digit. Instead, the entire 128 bits are shifted, and the 4 least significant bits are output.
Thus, when reading from Wave RAM, data might have changed its position. And, when writing to Wave RAM all data should be updated (it'd be no good idea to assume that old data is still located at the same position where it has been written to previously).

GBA Sound Channel 4 - Noise

This channel is used to output white noise. This is done by randomly switching the amplitude between high and low at a given frequency. Depending on the frequency the noise will appear 'harder' or 'softer'.

It is also possible to influence the function of the random generator, so the that the output becomes more regular, resulting in a limited ability to output Tone instead of Noise.

4000078h - SOUND4CNT_L (NR41, NR42) - Channel 4 Length/Envelope (R/W)

```
Bit       Expl.
0-5   W   Sound length; units of (64-n)/256s  (0-63)
6-7   -   Not used
8-10  R/W Envelope Step-Time; units of n/64s  (1-7, 0=No Envelope)
11    R/W Envelope Direction                  (0=Decrease, 1=Increase)
12-15 R/W Initial Volume of envelope          (1-15, 0=No Sound)
16-31 -   Not used
```

The Length value is used only if Bit 6 in NR44 is set.

400007Ch - SOUND4CNT_H (NR43, NR44) - Channel 4 Frequency/Control (R/W)
The amplitude is randomly switched between high and low at the given frequency. A higher frequency will make the noise to appear 'softer'.
When Bit 3 is set, the output will become more regular, and some frequencies will sound more like Tone than Noise.

```
  Bit        Expl.
  0-2   R/W  Dividing Ratio of Frequencies (r)
  3     R/W  Counter Step/Width (0=15 bits, 1=7 bits)
  4-7   R/W  Shift Clock Frequency (s)
  8-13  -    Not used
  14    R/W  Length Flag  (1=Stop output when length in NR41 expires)
  15    W    Initial      (1=Restart Sound)
  16-31 -    Not used
```
Frequency = 524288 Hz / r / 2^(s+1) ;For r=0 assume r=0.5 instead

Noise Random Generator (aka Polynomial Counter)
Noise randomly switches between HIGH and LOW levels, the output levels are calculated by a shift register (X), at the selected frequency, as such:
```
  7bit:  X=X SHR 1, IF carry THEN Out=HIGH, X=X XOR 60h ELSE Out=LOW
  15bit: X=X SHR 1, IF carry THEN Out=HIGH, X=X XOR 6000h ELSE Out=LOW
```
The initial value when (re-)starting the sound is X=40h (7bit) or X=4000h (15bit). The data stream repeats after 7Fh (7bit) or 7FFFh (15bit) steps.

GBA Sound Channel A and B - DMA Sound

The GBA contains two DMA sound channels (A and B), each allowing to replay digital sound (signed 8bit data, ie. -128..+127). Data can be transferred from INTERNAL memory (not sure if EXTERNAL memory works also ?) to FIFO by using DMA channel 1 or 2, the sample rate is generated by using one of the Timers.

40000A0h - FIFO_A_L - Sound A FIFO, Data 0 and Data 1 (W)
40000A2h - FIFO_A_H - Sound A FIFO, Data 2 and Data 3 (W)
These two registers may receive 32bit (4 bytes) of audio data (Data 0-3, Data 0 being located in least significant byte which is replayed first).
Internally, the capacity of the FIFO is 8 x 32bit (32 bytes), allowing to buffer a small amount of samples. As the name says (First In First Out), oldest data is replayed first.

40000A4h - FIFO_B_L - Sound B FIFO, Data 0 and Data 1 (W)
40000A6h - FIFO_B_H - Sound B FIFO, Data 2 and Data 3 (W)
Same as above, for Sound B.

Initializing DMA-Sound Playback
- Select Timer 0 or 1 in SOUNDCNT_H control register.
- Clear the FIFO.
- Manually write a sample byte to the FIFO.
- Initialize transfer mode for DMA 1 or 2.
- Initialize DMA Sound settings in sound control register.
- Start the timer.

DMA-Sound Playback Procedure
The pseudo-procedure below is automatically repeated.
```
  If Timer overflows then
    Move 8bit data from FIFO to sound circuit.
    If FIFO contains only 4 x 32bits (16 bytes) then
      Request more data per DMA
      Receive 4 x 32bit (16 bytes) per DMA
    Endif
  Endif
```
This playback mechanism will be repeated forever, regardless of the actual length of the sample buffer.

Synchronizing Sample Buffers

The buffer-end may be determined by counting sound Timer IRQs (each sample byte), or sound DMA IRQs (each 16th sample byte). Both methods would require a lot of CPU time (IRQ processing), and both would fail if interrupts are disabled for a longer period.
Better solutions would be to synchronize the sample rate/buffer length with V-blanks, or to use a second timer (in count up/slave mode) which produces an IRQ after the desired number of samples.

The Sample Rate
The GBA hardware does internally re-sample all sound output to 32.768kHz (default SOUNDBIAS setting). It'd thus do not make much sense to use higher DMA/Timer rates. Best re-sampling accuracy can be gained by using DMA/Timer rates of 32.768kHz, 16.384kHz, or 8.192kHz (ie. fragments of the physical output rate).

GBA Sound Control Registers

4000080h - SOUNDCNT_L (NR50, NR51) - Channel L/R Volume/Enable (R/W)

Bit		Expl.
0-2	R/W	Sound 1-4 Master Volume RIGHT (0-7)
3	-	Not used
4-6	R/W	Sound 1-4 Master Volume LEFT (0-7)
7	-	Not used
8-11	R/W	Sound 1-4 Enable Flags RIGHT (each Bit 8-11, 0=Disable, 1=Enable)
12-15	R/W	Sound 1-4 Enable Flags LEFT (each Bit 12-15, 0=Disable, 1=Enable)

4000082h - SOUNDCNT_H (GBA only) - DMA Sound Control/Mixing (R/W)

Bit		Expl.
0-1	R/W	Sound # 1-4 Volume (0=25%, 1=50%, 2=100%, 3=Prohibited)
2	R/W	DMA Sound A Volume (0=50%, 1=100%)
3	R/W	DMA Sound B Volume (0=50%, 1=100%)
4-7	-	Not used
8	R/W	DMA Sound A Enable RIGHT (0=Disable, 1=Enable)
9	R/W	DMA Sound A Enable LEFT (0=Disable, 1=Enable)
10	R/W	DMA Sound A Timer Select (0=Timer 0, 1=Timer 1)
11	W?	DMA Sound A Reset FIFO (1=Reset)
12	R/W	DMA Sound B Enable RIGHT (0=Disable, 1=Enable)
13	R/W	DMA Sound B Enable LEFT (0=Disable, 1=Enable)
14	R/W	DMA Sound B Timer Select (0=Timer 0, 1=Timer 1)
15	W?	DMA Sound B Reset FIFO (1=Reset)

4000084h - SOUNDCNT_X (NR52) - Sound on/off (R/W)
Bits 0-3 are automatically set when starting sound output, and are automatically cleared when a sound ends. (Ie. when the length expires, as far as length is enabled. The bits are NOT reset when an volume envelope ends.)

Bit		Expl.
0	R	Sound 1 ON flag (Read Only)
1	R	Sound 2 ON flag (Read Only)
2	R	Sound 3 ON flag (Read Only)
3	R	Sound 4 ON flag (Read Only)
4-6	-	Not used
7	R/W	PSG/FIFO Master Enable (0=Disable, 1=Enable) (Read/Write)
8-31	-	Not used

While Bit 7 is cleared, both PSG and FIFO sounds are disabled, and all PSG registers at 4000060h..4000081h are reset to zero (and must be re-initialized after re-enabling sound). However, registers 4000082h and 4000088h are kept read/write-able (of which, 4000082h has no function when sound is off, whilst 4000088h does work even when sound is off).

4000088h - SOUNDBIAS - Sound PWM Control (R/W, see below)
This register controls the final sound output. The default setting is 0200h, it is normally not required to change this value.

Bit	Expl.

```
0     -    Not used
1-9   R/W  Bias Level (Default=100h, converting signed samples into unsigned)
10-13 -    Not used
14-15 R/W  Amplitude Resolution/Sampling Cycle (Default=0, see below)
16-31 -    Not used
```
Amplitude Resolution/Sampling Cycle (0-3):
```
0  9bit / 32.768kHz   (Default, best for DMA channels A,B)
1  8bit / 65.536kHz
2  7bit / 131.072kHz
3  6bit / 262.144kHz  (Best for PSG channels 1-4)
```
For more information on this register, read the descriptions below.

400008Ch - Not used
400008Eh - Not used

Max Output Levels (with max volume settings)
Each of the two FIFOs can span the FULL output range (+/-200h).
Each of the four PSGs can span one QUARTER of the output range (+/-80h).
The current output levels of all six channels are added together by hardware.
So together, the FIFOs and PSGs, could reach THRICE the range (+/-600h).
The BIAS value is added to that signed value. With default BIAS (200h), the possible range becomes -400h..+800h, however, values that exceed the unsigned 10bit output range of 0..3FFh are clipped to MinMax(0,3FFh).

Resampling to 32.768kHz / 9bit (default)
The PSG channels 1-4 are internally generated at 262.144kHz, and DMA sound A-B could be theoretically generated at timer rates up to 16.78MHz. However, the final sound output is resampled to a rate of 32.768kHz, at 9bit depth (the above 10bit value, divided by two). If necessary, rates higher than 32.768kHz can be selected in the SOUNDBIAS register, that would result in a depth smaller than 9bit though.

PWM (Pulse Width Modulation) Output 16.78MHz / 1bit
Okay, now comes the actual output. The GBA can output only two voltages (low and high), these 'bits' are output at system clock speed (16.78MHz). If using the default 32.768kHz sampling rate, then 512 bits are output per sample (512*32K=16M). Each sample value (9bit range, N=0..511), would be then output as N low bits, followed by 512-N high bits. The resulting 'noise' is smoothed down by capacitors, by the speaker, and by human hearing, so that it will effectively sound like clean D/A converted 9bit voltages at 32kHz sampling rate.

Changing the BIAS Level
Normally use 200h for clean sound output. A value of 000h might make sense during periods when no sound is output (causing the PWM circuit to output low-bits only, which is eventually reducing the power consumption, and/or preventing 32KHz noise). Note: Using the SoundBias function (SWI 19h) allows to change the level by slowly incrementing or decrementing it (without hard scratch noise).

Low Power Mode
When not using sound output, power consumption can be reduced by setting both 4000084h (PSG/FIFO) and 4000088h (BIAS) to zero.

GBA Comparison of CGB and GBA Sound

The GBA sound controller is mostly the same than that of older monochrome gameboy and CGB. The following changes have been done:

New Sound Channels
Two new sound channels have been added that may be used to replay 8bit digital sound. Sample rate and sample

data must be supplied by using a Timer and a DMA channel.

New Control Registers
The SOUNDCNT_H register controls the new DMA channels - as well as mixing with the four old channels.
The SOUNDBIAS register controls the final sound output.

Sound Channel 3 Changes
The length of the Wave RAM is doubled by dividing it into two banks of 32 digits each, either one or both banks may be replayed (one after each other), for details check NR30 Bit 5-6. Optionally, the sound may be output at 75% volume, for details check NR32 Bit 7.

Changed Control Registers
NR50 is not supporting Vin signals (that's been an external sound input from cartridge).

Changed I/O Addresses
The GBAs sound register are located at 04000060-040000AE instead of at FF10-FF3F as in CGB and monochrome gameboy. However, note that there have been new blank spaces inserted between some of the separate registers - therefore it is NOT possible to port CGB software to GBA just by changing the sound base address.

Accessing I/O Registers
In some cases two of the old 8bit registers are packed into a 16bit register and may be accessed as such.

GBA Timers

The GBA includes four incrementing 16bit timers.
Timer 0 and 1 can be used to supply the sample rate for DMA sound channel A and/or B.

4000100h - TM0CNT_L - Timer 0 Counter/Reload (R/W)
4000104h - TM1CNT_L - Timer 1 Counter/Reload (R/W)
4000108h - TM2CNT_L - Timer 2 Counter/Reload (R/W)
400010Ch - TM3CNT_L - Timer 3 Counter/Reload (R/W)
Writing to these registers initializes the <reload> value (but does not directly affect the current counter value).
Reading returns the current <counter> value (or the recent/frozen counter value if the timer has been stopped).
The reload value is copied into the counter only upon following two situations: Automatically upon timer overflows, or when the timer start bit becomes changed from 0 to 1.
Note: When simultaneously changing the start bit from 0 to 1, and setting the reload value at the same time (by a single 32bit I/O operation), then the newly written reload value is recognized as new counter value.

4000102h - TM0CNT_H - Timer 0 Control (R/W)
4000106h - TM1CNT_H - Timer 1 Control (R/W)
400010Ah - TM2CNT_H - Timer 2 Control (R/W)
400010Eh - TM3CNT_H - Timer 3 Control (R/W)

```
Bit   Expl.
0-1   Prescaler Selection (0=F/1, 1=F/64, 2=F/256, 3=F/1024)
2     Count-up Timing   (0=Normal, 1=See below)  ;Not used in TM0CNT_H
3-5   Not used
6     Timer IRQ Enable  (0=Disable, 1=IRQ on Timer overflow)
7     Timer Start/Stop  (0=Stop, 1=Operate)
8-15  Not used
```

When Count-up Timing is enabled, the prescaler value is ignored, instead the time is incremented each time when the previous counter overflows. This function cannot be used for Timer 0 (as it is the first timer).
F = System Clock (16.78MHz).

GBA DMA Transfers

Overview
The GBA includes four DMA channels, the highest priority is assigned to DMA0, followed by DMA1, DMA2, and DMA3. DMA Channels with lower priority are paused until channels with higher priority have completed. The CPU is paused when DMA transfers are active, however, the CPU is operating during the periods when Sound/Blanking DMA transfers are paused.

Special features of the separate DMA channels
DMA0 - highest priority, best for timing critical transfers (eg. HBlank DMA).
DMA1 and DMA2 - can be used to feed digital sample data to the Sound FIFOs.
DMA3 - can be used to write to Game Pak ROM/FlashROM (but not GamePak SRAM).
Beside for that, each DMA 0-3 may be used for whatever general purposes.

40000B0h,0B2h - DMA0SAD - DMA 0 Source Address (W) (internal memory)
40000BCh,0BEh - DMA1SAD - DMA 1 Source Address (W) (any memory)
40000C8h,0CAh - DMA2SAD - DMA 2 Source Address (W) (any memory)
40000D4h,0D6h - DMA3SAD - DMA 3 Source Address (W) (any memory)
The most significant address bits are ignored, only the least significant 27 or 28 bits are used (max 07FFFFFFh internal memory, or max 0FFFFFFFh any memory - except SRAM ?!).

40000B4h,0B6h - DMA0DAD - DMA 0 Destination Address (W) (internal memory)
40000C0h,0C2h - DMA1DAD - DMA 1 Destination Address (W) (internal memory)
40000CCh,0CEh - DMA2DAD - DMA 2 Destination Address (W) (internal memory)
40000D8h,0DAh - DMA3DAD - DMA 3 Destination Address (W) (any memory)
The most significant address bits are ignored, only the least significant 27 or 28 bits are used (max. 07FFFFFFh internal memory or 0FFFFFFFh any memory - except SRAM ?!).

40000B8h - DMA0CNT_L - DMA 0 Word Count (W) (14 bit, 1..4000h)
40000C4h - DMA1CNT_L - DMA 1 Word Count (W) (14 bit, 1..4000h)
40000D0h - DMA2CNT_L - DMA 2 Word Count (W) (14 bit, 1..4000h)
40000DCh - DMA3CNT_L - DMA 3 Word Count (W) (16 bit, 1..10000h)
Specifies the number of data units to be transferred, each unit is 16bit or 32bit depending on the transfer type, a value of zero is treated as max length (ie. 4000h, or 10000h for DMA3).

40000BAh - DMA0CNT_H - DMA 0 Control (R/W)
40000C6h - DMA1CNT_H - DMA 1 Control (R/W)
40000D2h - DMA2CNT_H - DMA 2 Control (R/W)
40000DEh - DMA3CNT_H - DMA 3 Control (R/W)

```
Bit   Expl.
0-4   Not used
5-6   Dest Addr Control  (0=Increment,1=Decrement,2=Fixed,3=Increment/Reload)
7-8   Source Adr Control (0=Increment,1=Decrement,2=Fixed,3=Prohibited)
9     DMA Repeat                   (0=Off, 1=On) (Must be zero if Bit 11 set)
10    DMA Transfer Type            (0=16bit, 1=32bit)
11    Game Pak DRQ  - DMA3 only -  (0=Normal, 1=DRQ <from> Game Pak, DMA3)
12-13 DMA Start Timing  (0=Immediately, 1=VBlank, 2=HBlank, 3=Special)
        The 'Special' setting (Start Timing=3) depends on the DMA channel:
        DMA0=Prohibited, DMA1/DMA2=Sound FIFO, DMA3=Video Capture
14    IRQ upon end of Word Count   (0=Disable, 1=Enable)
15    DMA Enable                   (0=Off, 1=On)
```

After changing the Enable bit from 0 to 1, wait 2 clock cycles before accessing any DMA related registers.

When accessing OAM (7000000h) or OBJ VRAM (6010000h) by HBlank Timing, then the "H-Blank Interval Free" bit in DISPCNT register must be set.

Source and Destination Address and Word Count Registers
The SAD, DAD, and CNT_L registers are holding the initial start addresses, and initial length. The hardware does NOT change the content of these registers during or after the transfer.
The actual transfer takes place by using internal pointer/counter registers. The initial values are copied into internal regs under the following circumstances:
Upon DMA Enable (Bit 15) changing from 0 to 1: Reloads SAD, DAD, CNT_L.
Upon Repeat: Reloads CNT_L, and optionally DAD (Increment+Reload).

DMA Repeat bit
If the Repeat bit is cleared: The Enable bit is automatically cleared after the specified number of data units has been transferred.
If the Repeat bit is set: The Enable bit remains set after the transfer, and the transfer will be restarted each time when the Start condition (eg. HBlank, Fifo) becomes true. The specified number of data units is transferred <each> time when the transfer is (re-)started. The transfer will be repeated forever, until it gets stopped by software.

Sound DMA (FIFO Timing Mode) (DMA1 and DMA2 only)
In this mode, the DMA Repeat bit must be set, and the destination address must be FIFO_A (040000A0h) or FIFO_B (040000A4h).
Upon DMA request from sound controller, 4 units of 32bits (16 bytes) are transferred (both Word Count register and DMA Transfer Type bit are ignored). The destination address will not be incremented in FIFO mode.
Keep in mind that DMA channels of higher priority may offhold sound DMA. For example, when using a 64 kHz sample rate, 16 bytes of sound DMA data are requested each 0.25ms (4 kHz), at this time another 16 bytes are still in the FIFO so that there's still 0.25ms time to satisfy the DMA request. Thus DMAs with higher priority should not be operated for longer than 0.25ms. (This problem does not arise for HBlank transfers as HBlank time is limited to 16.212us.)

Game Pak DMA
Only DMA 3 may be used to transfer data to/from Game Pak ROM or Flash ROM - it cannot access Game Pak SRAM though (as SRAM data bus is limited to 8bit units). In normal mode, DMA is requested as long until Word Count becomes zero. When setting the 'Game Pack DRQ' bit, then the cartridge must contain an external circuit which outputs a /DREQ signal. Note that there is only one pin for /DREQ and /IREQ, thus the cartridge may not supply /IREQs while using DRQ mode.

Video Capture Mode (DMA3 only)
Intended to copy a bitmap from memory (or from external hardware/camera) to VRAM. When using this transfer mode, set the repeat bit, and write the number of data units (per scanline) to the word count register. Capture works similar like HBlank DMA, however, the transfer is started when VCOUNT=2, it is then repeated each scanline, and it gets stopped when VCOUNT=162.

Transfer End
The DMA Enable flag (Bit 15) is automatically cleared upon completion of the transfer. The user may also clear this bit manually in order to stop the transfer (obviously this is possible for Sound/Blanking DMAs only, in all other cases the CPU is stopped until the transfer completes by itself).

Transfer Rate/Timing
Except for the first data unit, all units are transferred by sequential reads and writes. For n data units, the DMA transfer time is:

```
2N+2(n-1)S+xI
```

Of which, 1N+(n-1)S are read cycles, and the other 1N+(n-1)S are write cycles, actual number of cycles depends on the waitstates and bus-width of the source and destination areas (as described in CPU Instruction Cycle

Times chapter). Internal time for DMA processing is 2I (normally), or 4I (if both source and destination are in gamepak memory area).

DMA lockup when stopping while starting ???
Capture delayed, Capture Enable=AutoCleared ???

GBA Communication Ports

The GBAs Serial Port may be used in various different communication modes. Normal mode may exchange data between two GBAs (or to transfer data from master GBA to several slave GBAs in one-way direction). Multi-player mode may exchange data between up to four GBAs. UART mode works much like a RS232 interface. JOY Bus mode uses a standardized Nintendo protocol. And General Purpose mode allows to mis-use the 'serial' port as bi-directional 4bit parallel port.
Note: The Nintendo DS does not include a Serial Port.

Wireless Adapter

Infrared Communication Adapters
Even though early GBA prototypes have been intended to support IR communication, this feature has been removed.
However, Nintendo is apparently considering to provide an external IR adapter (to be connected to the SIO connector, being accessed in General Purpose mode).
Also, it'd be theoretically possible to include IR ports built-in in game cartridges (as done for some older 8bit/monochrome Hudson games).

SIO Normal Mode

This mode is used to communicate between two units.
Transfer rates of 256Kbit/s or 2Mbit/s can be selected, however, the fast 2Mbit/s is intended ONLY for special hardware expansions that are DIRECTLY connected to the GBA link port (ie. without a cable being located between the GBA and expansion hardware). In normal cases, always use 256Kbit/s transfer rate which provides stable results.
Transfer lengths of 8bit or 32bit may be used, the 8bit mode is the same as for older DMG/CGB gameboys, however, the voltages for "GBA cartridges in GBAs" are different as for "DMG/CGB cartridges in DMG/CGB/GBAs", ie. it is not possible to communicate between DMG/CGB games and GBA games.

4000134h - RCNT (R) - Mode Selection, in Normal/Multiplayer/UART modes (R/W)

Bit	Expl.
0-3	Undocumented (current SC,SD,SI,SO state, as for General Purpose mode)
4-8	Not used (Should be 0, bits are read/write-able though)
9-13	Not used (Always 0, read only)
14	Not used (Should be 0, bit is read/write-able though)
15	Must be zero (0) for Normal/Multiplayer/UART modes

4000128h - SIOCNT - SIO Control, usage in NORMAL Mode (R/W)

```
Bit   Expl.
0     Shift Clock (SC)        (0=External, 1=Internal)
1     Internal Shift Clock    (0=256KHz, 1=2MHz)
2     SI State (opponents SO) (0=Low, 1=High/None) --- (Read Only)
3     SO during inactivity    (0=Low, 1=High) (applied ONLY when Bit7=0)
4-6   Not used                (Read only, always 0 ?)
7     Start Bit               (0=Inactive/Ready, 1=Start/Active)
8-11  Not used                (R/W, should be 0)
12    Transfer Length         (0=8bit, 1=32bit)
13    Must be "0" for Normal Mode
14    IRQ Enable              (0=Disable, 1=Want IRQ upon completion)
15    Not used                (Read only, always 0)
```

The Start bit is automatically reset when the transfer completes, ie. when all 8 or 32 bits are transferred, at that time an IRQ may be generated.

400012Ah - SIODATA8 - SIO Normal Communication 8bit Data (R/W)

For 8bit normal mode. Contains 8bit data (only lower 8bit are used). Outgoing data should be written to this register before starting the transfer. During transfer, transmitted bits are shifted-out (MSB first), and received bits are shifted-in simultaneously. Upon transfer completion, the register contains the received 8bit value.

4000120h - SIODATA32_L - SIO Normal Communication lower 16bit data (R/W)
4000122h - SIODATA32_H - SIO Normal Communication upper 16bit data (R/W)

Same as above SIODATA8, for 32bit normal transfer mode respectively.
SIOCNT/RCNT must be set to 32bit normal mode <before> writing to SIODATA32.

Initialization

First, initialize RCNT register. Second, set mode/clock bits in SIOCNT with startbit cleared. For master: select internal clock, and (in most cases) specify 256KHz as transfer rate. For slave: select external clock, the local transfer rate selection is then ignored, as the transfer rate is supplied by the remote GBA (or other computer, which might supply custom transfer rates).
Third, set the startbit in SIOCNT with mode/clock bits unchanged.

Recommended Communication Procedure for SLAVE unit (external clock)

- Initialize data which is to be sent to master.
- Set Start flag.
- Set SO to LOW to indicate that master may start now.
- Wait for IRQ (or for Start bit to become zero). (Check timeout here!)
- Set SO to HIGH to indicate that we are not ready.
- Process received data.
- Repeat procedure if more data is to be transferred.

(or is so=high done automatically? would be fine - more stable - otherwise master may still need delay)

Recommended Communication Procedure for SLAVE unit (external clock)

- Initialize data which is to be sent to master.
- Set Start=0 and SO=0 (SO=LOW indicates that slave is (almost) ready).
- Set Start=1 and SO=1 (SO=HIGH indicates not ready, applied after transfer).
  (Expl. Old SO=LOW kept output until 1st clock bit received).
  (Expl. New SO=HIGH is automatically output at transfer completion).
- Set SO to LOW to indicate that master may start now.
- Wait for IRQ (or for Start bit to become zero). (Check timeout here!)
- Process received data.
- Repeat procedure if more data is to be transferred.

Recommended Communication Procedure for MASTER unit (internal clock)

- Initialize data which is to be sent to slave.
- Wait for SI to become LOW (slave ready). (Check timeout here!)
- Set Start flag.
- Wait for IRQ (or for Start bit to become zero).
- Process received data.
- Repeat procedure if more data is to be transferred.

Cable Protocol
During inactive transfer, the shift clock (SC) is high. The transmit (SO) and receive (SI) data lines may be manually controlled as described above.
When master sends SC=LOW, each master and slave must output the next outgoing data bit to SO. When master sends SC=HIGH, each master and slave must read out the opponents data bit from SI. This is repeated for each of the 8 or 32 bits, and when completed SC will be kept high again.

Transfer Rates
Either 256KHz or 2MHz rates can be selected for SC, so max 32KBytes (256Kbit) or 128KBytes (2Mbit) can be transferred per second. However, the software must process each 8bit or 32bit of transmitted data separately, so the actual transfer rate will be reduced by the time spent on handling each data unit.
Only 256KHz provides stable results in most cases (such like when linking between two GBAs). The 2MHz rate is intended for special expansion hardware (with very short wires) only.

Using Normal mode for One-Way Multiplayer communication
When using normal mode with multiplay-cables, data isn't exchanged between first and second GBA as usually. Instead, data is shifted from first to last GBA (the first GBA receives zero, because master SI is shortcut to GND).
This behaviour may be used for fast ONE-WAY data transfer from master to all other GBAs. For example (3 GBAs linked):

```
Step         Sender      1st Recipient   2nd Recipient
Transfer 1:  DATA #0 --> UNDEF      -->  UNDEF     -->
Transfer 2:  DATA #1 --> DATA #0    -->  UNDEF     -->
Transfer 3:  DATA #2 --> DATA #1    -->  DATA #0   -->
Transfer 4:  DATA #3 --> DATA #2    -->  DATA #1   -->
```

The recipients should not output any own data, instead they should forward the previously received data to the next recipient during next transfer (just keep the incoming data unmodified in the data register).
Due to the delayed forwarding, 2nd recipient should ignore the first incoming data. After the last transfer, the sender must send one (or more) dummy data unit(s), so that the last data is forwarded to the 2nd (or further) recipient(s).

SIO Multi-Player Mode

Multi-Player mode can be used to communicate between up to 4 units.

4000134h - RCNT (R) - Mode Selection, in Normal/Multiplayer/UART modes (R/W)

```
Bit   Expl.
0-3   Undocumented (current SC,SD,SI,SO state, as for General Purpose mode)
4-8   Not used     (Should be 0, bits are read/write-able though)
9-13  Not used     (Always 0, read only)
14    Not used     (Should be 0, bit is read/write-able though)
15    Must be zero (0) for Normal/Multiplayer/UART modes
```

Note: Even though undocumented, many Nintendo games are using Bit 0 to test current SC state in multiplay mode.

4000128h - SIOCNT - SIO Control, usage in MULTI-PLAYER Mode (R/W)

```
Bit   Expl.
0-1   Baud Rate     (0-3: 9600,38400,57600,115200 bps)
```

```
  2     SI-Terminal   (0=Parent, 1=Child)                  (Read Only)
  3     SD-Terminal   (0=Bad connection, 1=All GBAs Ready) (Read Only)
  4-5   Multi-Player ID     (0=Parent, 1-3=1st-3rd child)  (Read Only)
  6     Multi-Player Error  (0=Normal, 1=Error)            (Read Only)
  7     Start/Busy Bit      (0=Inactive, 1=Start/Busy) (Read Only for Slaves)
  8-11  Not used            (R/W, should be 0)
  12    Must be "0" for Multi-Player mode
  13    Must be "1" for Multi-Player mode
  14    IRQ Enable          (0=Disable, 1=Want IRQ upon completion)
  15    Not used            (Read only, always 0)
```
The ID Bits are undefined until the first transfer has completed.

400012Ah - SIOMLT_SEND - Data Send Register (R/W)
Outgoing data (16 bit) which is to be sent to the other GBAs.

4000120h - SIOMULTI0 - SIO Multi-Player Data 0 (Parent) (R/W)
4000122h - SIOMULTI1 - SIO Multi-Player Data 1 (1st child) (R/W)
4000124h - SIOMULTI2 - SIO Multi-Player Data 2 (2nd child) (R/W)
4000126h - SIOMULTI3 - SIO Multi-Player Data 3 (3rd child) (R/W)
These registers are automatically reset to FFFFh upon transfer start.
After transfer, these registers contain incoming data (16bit each) from all remote GBAs (if any / otherwise still FFFFh), as well as the local outgoing SIOMLT_SEND data.
Ie. after the transfer, all connected GBAs will contain the same values in their SIOMULTI0-3 registers.

Initialization
- Initialize RCNT Bit 14-15 and SIOCNT Bit 12-13 to select Multi-Player mode.
- Read SIOCNT Bit 3 to verify that all GBAs are in Multi-Player mode.
- Read SIOCNT Bit 2 to detect whether this is the Parent/Master unit.

Recommended Transmission Procedure
- Write outgoing data to SIODATA_SEND.
- Master must set Start bit.
- All units must process received data in SIOMULTI0-3 when transfer completed.
- After the first successful transfer, ID Bits in SIOCNT are valid.
- If more data is to be transferred, repeat procedure.

The parent unit blindly sends data regardless of whether childs have already processed old data/supplied new data. So, parent unit might be required to insert delays between each transfer, and/or perform error checking. Also, slave units may signalize that they are not ready by temporarily switching into another communication mode (which does not output SD High, as Multi-Player mode does during inactivity).

Transfer Protocol
Beginning
- The masters SI pin is always LOW.
- When all GBAs are in Multiplayer mode (ready) SD is HIGH.
- When master starts the transfer, it sets SC=LOW, slaves receive Busy bit.

Step A
- ID Bits in master unit are set to 0.
- Master outputs Startbit (LOW), 16bit Data, Stopbit (HIGH) through SD.
- This data is written to SIOMULTI0 of all GBAs (including master).
- Master forwards LOW from its SO to 1st childs SI.
- Transfer ends if next child does not output data after certain time.

Step B
- ID Bits in 1st child unit are set to 1.
- 1st Child outputs Startbit (LOW), 16bit Data, Stopbit (HIGH) through SD.
- This data is written to SIOMULTI1 of all GBAs (including 1st child).
- 1st child forwards LOW from its SO to 2nd childs SI.

- Transfer ends if next child does not output data after certain time.
Step C
- ID Bits in 2nd child unit are set to 2.
- 2nd Child outputs Startbit (LOW), 16bit Data, Stopbit (HIGH) through SD.
- This data is written to SIOMULTI2 of all GBAs (including 2nd child).
- 2nd child forwards LOW from its SO to 3rd childs SI.
- Transfer ends if next child does not output data after certain time.
Step D
- ID Bits in 3rd child unit are set to 3.
- 3rd Child outputs Startbit (LOW), 16bit Data, Stopbit (HIGH) through SD.
- This data is written to SIOMULTI3 of all GBAs (including 3rd child).
- Transfer ends (this was the last child).
Transfer end
- Master sets SC=HIGH, all GBAs set SO=HIGH.
- The Start/Busy bits of all GBAs are automatically cleared.
- Interrupts are requested in all GBAs (as far as enabled).

Error Bit
This bit is set when a slave did not receive SI=LOW even though SC=LOW signalized a transfer (this might happen when connecting more than 4 GBAs, or when the previous child is not connected). Also, the bit is set when a Stopbit wasn't HIGH.
The error bit may be undefined during active transfer - read only after transfer completion (the transfer continues and completes as normal even if errors have occurred for some or all GBAs).
Don't know: The bit is automatically reset/initialized with each transfer, or must be manually reset?

Transmission Time
The transmission time depends on the selected Baud rate. And on the amount of Bits (16 data bits plus start/stop bits for each GBA), delays between data for each GBA, plus final timeout (if less than 4 GBAs). That is, depending on the number of connected GBAs:

GBAs	Bits	Delays	Timeout
1	18	None	Yes
2	36	1	Yes
3	54	2	Yes
4	72	3	None

(The average Delay and Timeout periods are unknown?)
Above is not counting the additional CPU time that must be spent on initiating and processing each transfer.

Fast One-Way Transmission
Beside for the actual SIO Multiplayer mode, you can also use SIO Normal mode for fast one-way data transfer from Master unit to all Child unit(s). See chapter about SIO Normal mode for details.

SIO UART Mode

This mode works much like a RS232 port, however, the voltages are unknown, probably 0/3V rather than +/-12V ?. SI and SO are data lines (with crossed wires), SC and SD signalize Clear to Send (with crossed wires also, which requires special cable when linking between two GBAs ?)

4000134h - RCNT (R) - Mode Selection, in Normal/Multiplayer/UART modes (R/W)

Bit	Expl.
0-3	Undocumented (current SC,SD,SI,SO state, as for General Purpose mode)
4-8	Not used (Should be 0, bits are read/write-able though)
9-13	Not used (Always 0, read only)
14	Not used (Should be 0, bit is read/write-able though)
15	Must be zero (0) for Normal/Multiplayer/UART modes

4000128h - SCCNT_L - SIO Control, usage in UART Mode (R/W)

```
Bit   Expl.
0-1   Baud Rate  (0-3: 9600,38400,57600,115200 bps)
2     CTS Flag   (0=Send always/blindly, 1=Send only when SC=LOW)
3     Parity Control (0=Even, 1=Odd)
4     Send Data Flag      (0=Not Full,  1=Full)    (Read Only)
5     Receive Data Flag   (0=Not Empty, 1=Empty)   (Read Only)
6     Error Flag          (0=No Error,  1=Error)   (Read Only)
7     Data Length         (0=7bits,   1=8bits)
8     FIFO Enable Flag    (0=Disable, 1=Enable)
9     Parity Enable Flag  (0=Disable, 1=Enable)
10    Send Enable Flag    (0=Disable, 1=Enable)
11    Receive Enable Flag (0=Disable, 1=Enable)
12    Must be "1" for UART mode
13    Must be "1" for UART mode
14    IRQ Enable          (0=Disable, 1=IRQ when any Bit 4/5/6 become set)
15    Not used            (Read only, always 0)
```

400012Ah - SIODATA8 - usage in UART Mode (R/W)

Addresses the send/receive shift register, or (when FIFO is used) the send/receive FIFO. In either case only the lower 8bit of SIODATA8 are used, the upper 8bit are not used.
The send/receive FIFO may store up to four 8bit data units each. For example, while 1 unit is still transferred from the send shift register, it is possible to deposit another 4 units in the send FIFO, which are then automatically moved to the send shift register one after each other.

Send/Receive Enable, CTS Feedback

The receiver outputs SD=LOW (which is input as SC=LOW at the remote side) when it is ready to receive data (that is, when Receive Enable is set, and the Receive shift register (or receive FIFO) isn't full.
When CTS flag is set to always/blindly, then the sender transmits data immediately when Send Enable is set, otherwise data is transmitted only when Send Enable is set and SC is LOW.

Error Flag

The error flag is set when a bad stop bit has been received (stop bit must be 0), when a parity error has occurred (if enabled), or when new data has been completely received while the receive data register (or receive FIFO) is already full.
The error flag is automatically reset when reading from SIOCNT register.

Init & Initback

The content of the FIFO is reset when FIFO is disabled in UART mode, thus, when entering UART mode initially set FIFO=disabled.
The Send/Receive enable bits must be reset before switching from UART mode into another SIO mode!

SIO JOY BUS Mode

This communication mode uses Nintendo's standardized JOY Bus protocol. When using this communication mode, the GBA is always operated as SLAVE!

In this mode, SI and SO pins are data lines (apparently synchronized by Start/Stop bits?), SC and SD are set to low (including during active transfer?), the transfer rate is unknown?

4000134h - RCNT (R) - Mode Selection, in JOY BUS mode (R/W)

```
Bit   Expl.
0-3   Undocumented (current SC,SD,SI,SO state, as for General Purpose mode)
4-8   Not used     (Should be 0, bits are read/write-able though)
9-13  Not used     (Always 0, read only)
```

```
14    Must be "1" for JOY BUS Mode
15    Must be "1" for JOY BUS Mode
```

4000128h - SIOCNT - SIO Control, not used in JOY BUS Mode
This register is not used in JOY BUS mode.

4000140h - JOYCNT - JOY BUS Control Register (R/W)
```
Bit   Expl.
0     Device Reset Flag     (Command FFh)          (Read/Acknowledge)
1     Receive Complete Flag (Command 14h or 15h?)  (Read/Acknowledge)
2     Send Complete Flag    (Command 15h or 14h?)  (Read/Acknowledge)
3-5   Not used
6     IRQ when receiving a Device Reset Command  (0=Disable, 1=Enable)
7-31  Not used
```
Bit 0-2 are working much like the bits in the IF register: Write a "1" bit to reset (acknowledge) the respective bit.
UNCLEAR: Interrupts can be requested for Send/Receive commands also?

4000150h - JOY_RECV_L - Receive Data Register low (R/W)
4000152h - JOY_RECV_H - Receive Data Register high (R/W)
4000154h - JOY_TRANS_L - Send Data Register low (R/W)
4000156h - JOY_TRANS_H - Send Data Register high (R/W)
Send/receive data registers.

4000158h - JOYSTAT - Receive Status Register (R/W)
```
Bit   Expl.
0     Not used
1     Receive Status Flag   (0=Remote GBA is/was receiving) (Read Only?)
2     Not used
3     Send Status Flag      (1=Remote GBA is/was sending)   (Read Only?)
4-5   General Purpose Flag  (Not assigned, may be used for whatever purpose)
6-31  Not used
```
Bit 1 is automatically set when writing to local JOY_TRANS.
Bit 3 is automatically reset when reading from local JOY_RECV.

Below are the four possible commands which can be received by the GBA. Note that the GBA (slave) cannot send any commands itself, all it can do is to read incoming data, and to provide 'reply' data which may (or may not) be read out by the master unit.

Command FFh - Device Reset
```
Receive FFh (Command)
Send    00h (GBA Type number LSB (or MSB?))
Send    04h (GBA Type number MSB (or LSB?))
Send    XXh (lower 8bits of SIOSTAT register)
```

Command 00h - Type/Status Data Request
```
Receive 00h (Command)
Send    00h (GBA Type number LSB (or MSB?))
Send    04h (GBA Type number MSB (or LSB?))
Send    XXh (lower 8bits of SIOSTAT register)
```

Command 15h - GBA Data Write (to GBA)
```
Receive 15h (Command)
Receive XXh (Lower 8bits of JOY_RECV_L)
Receive XXh (Upper 8bits of JOY_RECV_L)
Receive XXh (Lower 8bits of JOY_RECV_H)
Receive XXh (Upper 8bits of JOY_RECV_H)
Send    XXh (lower 8bits of SIOSTAT register)
```

Command 14h - GBA Data Read (from GBA)

```
Receive 14h (Command)
Send    XXh (Lower 8bits of JOY_TRANS_L)
Send    XXh (Upper 8bits of JOY_TRANS_L)
Send    XXh (Lower 8bits of JOY_TRANS_H)
Send    XXh (Upper 8bits of JOY_TRANS_H)
Send    XXh (lower 8bits of SIOSTAT register)
```

SIO General-Purpose Mode

In this mode, the SIO is 'misused' as a 4bit bi-directional parallel port, each of the SI,SO,SC,SD pins may be directly controlled, each can be separately declared as input (with internal pull-up) or as output signal.

4000134h - RCNT (R) - SIO Mode, usage in GENERAL-PURPOSE Mode (R/W)
Interrupts can be requested when SI changes from HIGH to LOW, as General Purpose mode does not require a serial shift clock, this interrupt may be produced even when the GBA is in Stop (low power standby) state.

```
Bit   Expl.
0     SC Data Bit         (0=Low, 1=High)
1     SD Data Bit         (0=Low, 1=High)
2     SI Data Bit         (0=Low, 1=High)
3     SO Data Bit         (0=Low, 1=High)
4     SC Direction        (0=Input, 1=Output)
5     SD Direction        (0=Input, 1=Output)
6     SI Direction        (0=Input, 1=Output, but see below)
7     SO Direction        (0=Input, 1=Output)
8     SI Interrupt Enable (0=Disable, 1=Enable)
9-13  Not used
14    Must be "0" for General-Purpose Mode
15    Must be "1" for General-Purpose or JOYBUS Mode
```

SI should be always used as Input to avoid problems with other hardware which does not expect data to be output there.

4000128h - SIOCNT - SIO Control, not used in GENERAL-PURPOSE Mode
This register is not used in general purpose mode. That is, the separate bits of SIOCNT still exist and are read- and/or write-able in the same manner as for Normal, Multiplay, or UART mode (depending on SIOCNT Bit 12,13), but are having no effect on data being output to the link port.

SIO Control Registers Summary

Mode Selection (by RCNT.15-14 and SIOCNT.13-12)

```
R.15 R.14 S.13 S.12 Mode
0    x    0    0    Normal 8bit
0    x    0    1    Normal 32bit
0    x    1    0    Multiplay 16bit
0    x    1    1    UART (RS232)
1    0    x    x    General Purpose
1    1    x    x    JOY BUS
```

SIOCNT

```
Bit    0      1    2     3      4 5 6   7     8    9      10   11
Normal Master Rate SI/In SO/Out - - -   Start -    -      -    -
Multi  Baud   Baud SI/In SD/In  ID# Err Start -    -      -    -
UART   Baud   Baud CTS   Parity S R Err Bits  FIFO Parity Send Recv
```

GBA Wireless Adapter

GBA Wireless Adapter (AGB-015 or OXY-004)

GBA Wireless Adapter Games

GBA Wireless Adapter compatible Games

```
bit Generations series (Japan only)
Boktai 2: Solar Boy Django (Konami)
Boktai 3: Sabata's Counterattack
Classic NES Series: Donkey Kong
Classic NES Series: Dr. Mario
Classic NES Series: Ice Climber
Classic NES Series: Pac-Man
Classic NES Series: Super Mario Bros.
Classic NES Series: Xevious
Digimon Racing (Bandai) (No Wireless Adapter support in European release)
Dragon Ball Z: Buu's Fury (Atari)
Famicom Mini Series: #13 Balloon Fight
Famicom Mini Series: #12 Clu Clu Land
Famicom Mini Series: #16 Dig Dug
Famicom Mini Series: #02 Donkey Kong
Famicom Mini Series: #15 Dr. Mario
Famicom Mini Series: #03 Ice Climber
Famicom Mini Series: #18 Makaimura
Famicom Mini Series: #08 Mappy
Famicom Mini Series: #11 Mario Bros.
Famicom Mini Series: #06 Pac-Man
Famicom Mini Series: #30 SD Gundam World Scramble Wars
Famicom Mini Series: #01 Super Mario Bros.
Famicom Mini Series: #21 Super Mario Bros.
Famicom Mini Series: #19 Twin Bee
Famicom Mini Series: #14 Wrecking Crew
Famicom Mini Series: #07 Xevious
Hamtaro: Ham-Ham Games (Nintendo)
Lord of the Rings: The Third Age, The (EA Games)
Mario Golf: Advance Tour (Nintendo)
Mario Tennis: Power Tour (Nintendo)
Mega Man Battle Network 5: Team Protoman (Capcom)
Mega Man Battle Network 5: Team Colonel (Capcom)
Mega Man Battle Network 6: Cybeast Falzar
Mega Man Battle Network 6: Cybeast Gregar
Momotaro Dentetsu G: Make a Gold Deck! (Japan only)
Pokemon Emerald (Nintendo)
Pokemon FireRed (Nintendo)
Pokemon LeafGreen (Nintendo)
Sennen Kazoku (Japan only)
Shrek SuperSlam
Sonic Advance 3
```

GBA Wireless Adapter Login

GBA Wireless Adapter Login

```
rcnt=8000h    ;\
rcnt=80A0h    ;
```

```
 rcnt=80A2h    ; reset adapter or so
 wait          ;
 rcnt=80A0h    ;/
 siocnt=5003h  ;\set 32bit normal mode, 2MHz internal clock
 rcnt=0000h    ;/
 passes=0, index=0
@@lop:
 passes=passes+1, if passes>32 then ERROR  ;give up (usually only 10 passses)
 recv.lo=siodata AND FFFFh    ;response from adapter
 recv.hi=siodata/10000h       ;adapter's own "NI" data
 if send.hi<>recv.lo then index=0, goto @@stuck  ;<-- fallback to index=0
 if (send.lo XOR FFFFh)<>recv.lo then goto @@stuck
 if (send.hi XOR FFFFh)<>recv.hi then goto @@stuck
 index=index+1
@@stuck:
 send.lo=halfword[@@key_string+index*2]
 send.hi=recv.hi XOR FFFFh
 siodata=send.lo+(send.hi*10000h)
 siocnt.bit7=1                        ;<-- start transmission
 if index<4 then goto @@lop
 ret
@@key_string db 'NINTENDO',01h,80h    ;10 bytes (5 halfwords; index=0..4)
```

Data exchanged during Login

```
              GBA                          ADAPTER
              xxxx494E ;\     <-->         xxxxxxxx
              xxxx494E ; "NI" <--> "NI"/; 494EB6B1 ;\
 NOT("NI") /; B6B1494E ;/     <-->     \; 494EB6B1 ; NOT("NI")
           \; B6B1544E ;\"NT" <--> "NT"/; 544EB6B1 ;/
 NOT("NT") /; ABB1544E ;/     <-->     \; 544EABB1 ;\NOT("NT")
           \; ABB14E45 ;\"EN" <--> "EN"/; 4E45ABB1 ;/
 NOT("EN") /; B1BA4E45 ;/     <-->     \; 4E45B1BA ;\NOT("EN")
           \; B1BA4F44 ;\"DO" <--> "DO"/; 4F44B1BA ;/
 NOT("DO") /; B0BB4F44 ;/     <-->     \; 4F44B0BB ;\NOT("DO")
           \; B0BB8001 ;-fin  <-->  fin-; 8001B0BB ;/
                \   \                        \   \
                 \   LSBs=Own                 \   LSBs=Inverse of
                  \   Data.From.Gba            \   Prev.Data.From.Gba
                   \                            \
                    MSBs=Inverse of              MSBs=Own
                     Prev.Data.From.Adapter       Data.From.Adapter
```

GBA Wireless Adapter Commands

Wireless Command/Parameter Transmission

```
 GBA       Adapter
 9966ppcch 80000000h   ;-send command (cc), and num param_words (pp)
 <param01> 80000000h   ;\
 <param02> 80000000h   ; send "pp" parameter word(s), if any
 ...       ...         ;/
 80000000h 9966rraah   ;-recv ack (aa=cc+80h), and num response_words (rr)
 80000000? <reply01>   ;\
 80000000? <reply02>   ; recv "rr" response word(s), if any
 ...       ...         ;/
```

Wireless 32bit Transfers

```
 wait until [4000128h].Bit2=0  ;want SI=0
 set [4000128h].Bit3=1         ;set SO=1
 wait until [4000128h].Bit2=1  ;want SI=1
 set [4000128h].Bit3=0,Bit7=1  ;set SO=0 and start 32bit transfer
```

All command/param/reply transfers should be done at Internal Clock (except, Response Words for command 25h,27h,35h,37h should use External Clock).

Wireless Commands

```
  Cmd Para Reply Name
  10h -    -     Hello (send immediately after login)
  11h -    1     Good/Bad response to cmd 16h ?
  12h
  13h -    1
  14h
  15h
  16h 6    -     Introduce (send game/user name)
  17h 1    -     Config (send after Hello) (eg. param=003C0420h or 003C043Ch)
  18h
  19h
  1Ah
  1Bh
  1Ch -    -
  1Dh -    NN    Get Directory? (receive list of game/user names?)
  1Eh -    NN    Get Directory? (receive list of game/user names?)
  1Fh 1    -     Select Game for Download (send 16bit Game_ID)

  20h -    1
  21h -    1     Good/Bad response to cmd 1Fh ?
  22h
  23h
  24h -    -
  25h                                  ;use EXT clock!
  26h -    -
  27h -    -     Begin Download ?      ;use EXT clock!
  28h
  29h
  2Ah
  2Bh
  2Ch
  2Dh
  2Eh
  2Fh

  30h 1    -
  31h
  32h
  33h
  34h
  35h                                  ;use EXT clock!
  36h
  37h                                  ;use EXT clock!
  38h
  39h
  3Ah
  3Bh
  3Ch
  3Dh -    -     Bye (return to language select)
  3Eh
  3Fh
```

Special Response 996601EEh for error or so? (only at software side?)

GBA Wireless Adapter Component Lists

Main Chipset

```
  U1 32pin Freescale MC13190 (2.4 GHz ISM band transceiver)
  U2 48pin Freescale CT3000 or CT3001 (depending on adapter version)
  X3  2pin 9.5MHz crystal
```

The MC13190 is a Short-Range, Low-Power 2.4 GHz ISM band transceiver.
The processor is Motorola's 32-bit M-Core RISC engine. (?) MCT3000 (?)
See also: http://www.eetimes.com/document.asp?doc_id=1271943

Version with GERMAN Postal Code on sticker:

```
  Sticker on Case:
    "GAME BOY advance, WIRELESS ADAPTER"
    "Pat.Pend.Made in Philipines, CE0125(!)B"
    "MODEL NO./MODELE NO.AGB-015 D-63760 Grossosteim P/AGB-A-WA-EUR-2 E3"
  PCB: "19-C046-04, A-7" (top side) and "B-7" and Microchip ",\\" (bottom side)
  PCB: white stamp "3104, 94V-0, RU, TW-15"
  PCB: black stamp "22FDE"
  U1 32pin "Freescale 13190, 4WFQ" (MC13190) (2.4 GHz ISM band transceiver)
  U2 48pin "Freescale CT3001, XAC0445"  (bottom side)
  X3  2pin "D959L4I" (9.5MHz)           (top side) (ca. 19 clks per 2us)
```

Further components... top side (A-7)

```
  D1   5pin "D6F, 44"   (top side, below X3)
  U71  6pin ".., () 2"  (top side, right of X3, tiny black chip)
  B71  6pin "[]"        (top side, right of X3, small white chip)
  ANT  2pin on-board copper wings
  Q?   3pin             (top side, above CN1)
  Q?   3pin             (top side, above CN1)
  D?   2pin "72"        (top side, above CN1)
  D3   2pin "F2"        (top side, above CN1)
  U200 4pin "MSV"       (top side, above CN1)
  U202 5pin "LXKA"      (top side, right of CN1)
  U203 4pin "M6H"       (top side, right of CN1)
  CN1  6pin connector to GBA link port (top side)
```

Further components... bottom side (B-7)

```
  U201 5pin "LXVB"      (bottom side, near CN1)
  U72  4pin "BMs"       (bottom side, near ANT, tiny black chip)
  FL70 ?pin "[] o26"    (bottom side, near ANT, bigger white chip)
  B70  6pin "[]"        (bottom side, near ANT, small white chip)
```

Plus, resistors and capacitors (without any markings).

Version WITHOUT sticker:

```
  Sticker on Case: N/A
  PCB: "19-C046-03, A-1" (top side) and "B-1" and Microchip ",\\" (bottom side)
  PCB: white stamp "3204, TW-15, RU, 94V-0"
  PCB: black stamp "23MN" or "23NH" or so (smeared)
  U1 32pin "Freescale 13190, 4FGD"      (top side)
  U2 48pin "Freescale CT3000, XAB0425"  (bottom side) ;CT3000 (not CT3001)
  X3  2pin "9.5SKSS4GT"                 (top side)
```

Further components... top side (A-1)

```
  D1   5pin "D6F, 31"   (top side, below X3)
  U71  6pin "P3, () 2"  (top side, right of X3, tiny black chip)
  B71  6pin "[]"        (top side, right of X3, small white chip)
  ANT  2pin on-board copper wings
  Q70  3pin             (top side, above CN1)
  D?   2pin "72"        (top side, above CN1)
  D3   2pin "F2"        (top side, above CN1)
  U200 4pin "MSV"       (top side, above CN1)
  U202 5pin "LXKH"      (top side, right of CN1)
  U203 4pin "M6H"       (top side, right of CN1)
  CN1  6pin connector to GBA link port (top side)
```

Further components... bottom side (B-1)

```
  U201 5pin "LXV2"      (bottom side, near CN1)
  U70  6pin "AAG"       (bottom side, near ANT, tiny black chip)
  FL70 ?pin "[] o26"    (bottom side, near ANT, bigger white chip)
  B70  6pin "[]"        (bottom side, near ANT, small white chip)
```

Plus, resistors and capacitors (without any markings).

Major Differences

```
Sticker      "N/A"                    vs "Grossosteim P/AGB-A-WA-EUR-2 E3"
PCB-markings "19-C046-03, A-1, 3204"  vs "19-C046-04, A-7, 3104"
U1           "CT3000, XAB0425"        vs "CT3001, XAC0445"
Transistors  One transistor (Q70)     vs Two transistors (both nameless)
U70/U72      U70 "AAG" (6pin)         vs U72 "BMs" (4pin)
```

Purpose of the changes is unknown (either older/newer revisions, or different regions with different FCC regulations).

GBA Infrared Communication

Early GBA prototypes have been intended to include a built-in IR port for sending and receiving IR signals. Among others, this port could have been used to communicate with other GBAs, or older CGB models, or TV Remote Controls, etc.

[THE INFRARED COMMUNICATION FEATURE IS -NOT- SUPPORTED ANYMORE]
Anyways, the prototype specifications have been as shown below...

Keep in mind that the IR signal may be interrupted by whatever objects moved between sender and receiver - the IR port isn't recommended for programs that require realtime data exchange (such like action games).

4000136h - IR - Infrared Register (R/W)

```
Bit   Expl.
0     Transmission Data  (0=LED Off, 1=LED On)
1     READ Enable        (0=Disable, 1=Enable)
2     Reception Data     (0=None, 1=Signal received) (Read only)
3     AMP Operation      (0=Off, 1=On)
4     IRQ Enable Flag    (0=Disable, 1=Enable)
5-15  Not used
```

When IRQ is enabled, an interrupt is requested if the incoming signal was 0.119us Off (2 cycles), followed by 0.536us On (9 cycles) - minimum timing periods each.

Transmission Notes
When transmitting an IR signal, note that it'd be not a good idea to keep the LED turned On for a very long period (such like sending a 1 second synchronization pulse). The recipient's circuit would treat such a long signal as "normal IR pollution which is in the air" after a while, and thus ignore the signal.

Reception Notes
Received data is internally latched. Latched data may be read out by setting both READ and AMP bits.
Note: Provided that you don't want to receive your own IR signal, be sure to set Bit 0 to zero before attempting to receive data.

Power-consumption
After using the IR port, be sure to reset the register to zero in order to reduce battery power consumption.

GBA Keypad Input

The built-in GBA gamepad has 4 direction keys, and 6 buttons.

4000130h - KEYINPUT - Key Status (R)

```
Bit   Expl.
0     Button A       (0=Pressed, 1=Released)
1     Button B       (etc.)
2     Select         (etc.)
```

```
  3     Start           (etc.)
  4     Right           (etc.)
  5     Left            (etc.)
  6     Up              (etc.)
  7     Down            (etc.)
  8     Button R        (etc.)
  9     Button L        (etc.)
  10-15 Not used
```
It'd be usually recommended to read-out this register only once per frame, and to store the current state in memory. As a side effect, this method avoids problems caused by switch bounce when a key is newly released or pressed.

4000132h - KEYCNT - Key Interrupt Control (R/W)
The keypad IRQ function is intended to terminate the very-low-power Stop mode, it is not suitable for processing normal user input, to do this, most programs are invoking their keypad handlers from within VBlank IRQ.
```
  Bit   Expl.
  0     Button A        (0=Ignore, 1=Select)
  1     Button B        (etc.)
  2     Select          (etc.)
  3     Start           (etc.)
  4     Right           (etc.)
  5     Left            (etc.)
  6     Up              (etc.)
  7     Down            (etc.)
  8     Button R        (etc.)
  9     Button L        (etc.)
  10-13 Not used
  14    IRQ Enable Flag (0=Disable, 1=Enable)
  15    IRQ Condition   (0=Logical OR, 1=Logical AND)
```
In logical OR mode, an interrupt is requested when at least one of the selected buttons is pressed.
In logical AND mode, an interrupt is requested when ALL of the selected buttons are pressed.

Notes
In 8bit gameboy compatibility mode, L and R Buttons are used to toggle the screen size between normal 160x144 pixels and stretched 240x144 pixels.
The GBA SP is additionally having a * Button used to toggle the backlight on and off (controlled by separate hardware logic, there's no way to detect or change the current backlight state by software).

GBA Interrupt Control

4000208h - IME - Interrupt Master Enable Register (R/W)
```
  Bit   Expl.
  0     Disable all interrupts        (0=Disable All, 1=See IE register)
  1-31  Not used
```

4000200h - IE - Interrupt Enable Register (R/W)
```
  Bit   Expl.
  0     LCD V-Blank                   (0=Disable)
  1     LCD H-Blank                   (etc.)
  2     LCD V-Counter Match           (etc.)
  3     Timer 0 Overflow              (etc.)
  4     Timer 1 Overflow              (etc.)
  5     Timer 2 Overflow              (etc.)
  6     Timer 3 Overflow              (etc.)
  7     Serial Communication          (etc.)
  8     DMA 0                         (etc.)
  9     DMA 1                         (etc.)
  10    DMA 2                         (etc.)
```

```
  11    DMA 3                          (etc.)
  12    Keypad                         (etc.)
  13    Game Pak (external IRQ source) (etc.)
  14-15 Not used
```
Note that there is another 'master enable flag' directly in the CPUs Status Register (CPSR) accessible in privileged modes, see CPU reference for details.

4000202h - IF - Interrupt Request Flags / IRQ Acknowledge (R/W, see below)
```
  Bit   Expl.
  0     LCD V-Blank                    (1=Request Interrupt)
  1     LCD H-Blank                    (etc.)
  2     LCD V-Counter Match            (etc.)
  3     Timer 0 Overflow               (etc.)
  4     Timer 1 Overflow               (etc.)
  5     Timer 2 Overflow               (etc.)
  6     Timer 3 Overflow               (etc.)
  7     Serial Communication           (etc.)
  8     DMA 0                          (etc.)
  9     DMA 1                          (etc.)
  10    DMA 2                          (etc.)
  11    DMA 3                          (etc.)
  12    Keypad                         (etc.)
  13    Game Pak (external IRQ source) (etc.)
  14-15 Not used
```
Interrupts must be manually acknowledged by writing a "1" to one of the IRQ bits, the IRQ bit will then be cleared.

"[Cautions regarding clearing IME and IE]
A corresponding interrupt could occur even while a command to clear IME or each flag of the IE register is being executed. When clearing a flag of IE, you need to clear IME in advance so that mismatching of interrupt checks will not occur." ?

"[When multiple interrupts are used]
When the timing of clearing of IME and the timing of an interrupt agree, multiple interrupts will not occur during that interrupt. Therefore, set (enable) IME after saving IME during the interrupt routine." ?

BIOS Interrupt handling
Upon interrupt execution, the CPU is switched into IRQ mode, and the physical interrupt vector is called - as this address is located in BIOS ROM, the BIOS will always execute the following code before it forwards control to the user handler:
```
  00000018  b      128h                ;IRQ vector: jump to actual BIOS handler
  00000128  stmfd  r13!,r0-r3,r12,r14  ;save registers to SP_irq
  0000012C  mov    r0,4000000h         ;ptr+4 to 03FFFFFC (mirror of 03007FFC)
  00000130  add    r14,r15,0h          ;retadr for USER handler $+8=138h
  00000134  ldr    r15,[r0,-4h]        ;jump to [03FFFFFC] USER handler
  00000138  ldmfd  r13!,r0-r3,r12,r14  ;restore registers from SP_irq
  0000013C  subs   r15,r14,4h          ;return from IRQ (PC=LR-4, CPSR=SPSR)
```
As shown above, a pointer to the 32bit/ARM-code user handler must be setup in [03007FFCh]. By default, 160 bytes of memory are reserved for interrupt stack at 03007F00h-03007F9Fh.

Recommended User Interrupt handling
- If necessary switch to THUMB state manually (handler is called in ARM state)
- Determine reason(s) of interrupt by examining IF register
- User program may freely assign priority to each reason by own logic
- Process the most important reason of your choice
- User MUST manually acknowledge by writing to IF register
- If user wants to allow nested interrupts, save SPSR_irq, then enable IRQs.
- If using other registers than BIOS-pushed R0-R3, manually save R4-R11 also.

- Note that Interrupt Stack is used (which may have limited size)
- So, for memory consuming stack operations use system mode (=user stack).
- When calling subroutines in system mode, save LSR_usr also.
- Restore SPSR_irq and/or R4-R11 if you've saved them above.
- Finally, return to BIOS handler by BX LR (R14_irq) instruction.

Default memory usage at 03007FXX (and mirrored to 03FFFFXX)

```
Addr.    Size Expl.
3007FFCh 4    Pointer to user IRQ handler (32bit ARM code)
3007FF8h 2    Interrupt Check Flag (for IntrWait/VBlankIntrWait functions)
3007FF4h 4    Allocated Area
3007FF0h 4    Pointer to Sound Buffer
3007FE0h 16   Allocated Area
3007FA0h 64   Default area for SP_svc Supervisor Stack (4 words/time)
3007F00h 160  Default area for SP_irq Interrupt Stack (6 words/time)
```

Memory below 7F00h is free for User Stack and user data. The three stack pointers are initially initialized at the TOP of the respective areas:

```
SP_svc=03007FE0h
SP_irq=03007FA0h
SP_usr=03007F00h
```

The user may redefine these addresses and move stacks into other locations, however, the addresses for system data at 7FE0h-7FFFh are fixed.

Not sure, is following free for user ?

Registers R8-R12_fiq, R13_fiq, R14_fiq, SPSR_fiq
Registers R13-R14_abt, SPSR_abt
Registers R13-R14_und, SPSR_und

Fast Interrupt (FIQ)

The ARM CPU provides two interrupt sources, IRQ and FIQ. In the GBA only IRQ is used. In normal GBAs, the FIQ signal is shortcut to VDD35, ie. the signal is always high, and there is no way to generate a FIQ by hardware. The registers R8..12_fiq could be used by software (when switching into FIQ mode by writing to CPSR) - however, this might make the game incompatible with hardware debuggers (which are reportedly using FIQs for debugging purposes).

GBA System Control

4000204h - WAITCNT - Waitstate Control (R/W)

This register is used to configure game pak access timings. The game pak ROM is mirrored to three address regions at 08000000h, 0A000000h, and 0C000000h, these areas are called Wait State 0-2. Different access timings may be assigned to each area (this might be useful in case that a game pak contains several ROM chips with different access times each).

```
Bit   Expl.
0-1   SRAM Wait Control          (0..3 = 4,3,2,8 cycles)
2-3   Wait State 0 First Access  (0..3 = 4,3,2,8 cycles)
4     Wait State 0 Second Access (0..1 = 2,1 cycles)
5-6   Wait State 1 First Access  (0..3 = 4,3,2,8 cycles)
7     Wait State 1 Second Access (0..1 = 4,1 cycles; unlike above WS0)
8-9   Wait State 2 First Access  (0..3 = 4,3,2,8 cycles)
10    Wait State 2 Second Access (0..1 = 8,1 cycles; unlike above WS0,WS1)
11-12 PHI Terminal Output        (0..3 = Disable, 4.19MHz, 8.38MHz, 16.78MHz)
13    Not used
14    Game Pak Prefetch Buffer (Pipe) (0=Disable, 1=Enable)
15    Game Pak Type Flag  (Read Only) (0=GBA, 1=CGB) (IN35 signal)
16-31 Not used
```

At startup, the default setting is 0000h. Currently manufactured cartridges are using the following settings: WS0/ROM=3,1 clks; SRAM=8 clks; WS2/EEPROM: 8,8 clks; prefetch enabled; that is, WAITCNT=4317h, for more info see "GBA Cartridges" chapter.

First Access (Non-sequential) and Second Access (Sequential) define the waitstates for N and S cycles, the actual access time is 1 clock cycle PLUS the number of waitstates.
GamePak uses 16bit data bus, so that a 32bit access is split into TWO 16bit accesses (of which, the second fragment is always sequential, even if the first fragment was non-sequential).

GBA GamePak Prefetch

NOTES:
The GBA forcefully uses non-sequential timing at the beginning of each 128K-block of gamepak ROM, eg. "LDMIA [801fff8h],r0-r7" will have non-sequential timing at 8020000h.
The PHI Terminal output (PHI Pin of Gamepak Bus) should be disabled.

4000300h - POSTFLG - BYTE - Undocumented - Post Boot / Debug Control (R/W)
After initial reset, the GBA BIOS initializes the register to 01h, and any further execution of the Reset vector (00000000h) will pass control to the Debug vector (0000001Ch) when sensing the register to be still set to 01h.

```
Bit   Expl.
0     Undocumented. First Boot Flag  (0=First, 1=Further)
1-7   Undocumented. Not used.
```

Normally the debug handler rejects control unless it detects Debug flags in cartridge header, in that case it may redirect to a cut-down boot procedure (bypassing Nintendo logo and boot delays, much like nocash burst boot for multiboot software). I am not sure if it is possible to reset the GBA externally without automatically resetting register 300h though.

4000301h - HALTCNT - BYTE - Undocumented - Low Power Mode Control (W)
Writing to this register switches the GBA into battery saving mode.
In Halt mode, the CPU is paused as long as (IE AND IF)=0, this should be used to reduce power-consumption during periods when the CPU is waiting for interrupt events.
In Stop mode, most of the hardware including sound and video are paused, this very-low-power mode could be used much like a screensaver.

```
Bit   Expl.
0-6   Undocumented. Not used.
7     Undocumented. Power Down Mode  (0=Halt, 1=Stop)
```

The current GBA BIOS addresses only the upper eight bits of this register (by writing 00h or 80h to address 04000301h), however, as the register isn't officially documented, some or all of the bits might have different meanings in future GBA models.
For best forwards compatibility, it'd generally be more recommended to use the BIOS Functions SWI 2 (Halt) or SWI 3 (Stop) rather than writing to this register directly.

4000410h - Undocumented - Purpose Unknown ? 8bit (W)
The BIOS writes the 8bit value 0FFh to this address. Purpose Unknown.
Probably just another bug in the BIOS.

4000800h - 32bit - Undocumented - Internal Memory Control (R/W)
Supported by GBA and GBA SP only - NOT supported by DS (even in GBA mode).
Also supported by GBA Micro - but crashes on "overclocked" WRAM setting.
Initialized to 0D000020h (by hardware). Unlike all other I/O registers, this register is mirrored across the whole I/O area (in increments of 64K, ie. at 4000800h, 4010800h, 4020800h, ..., 4FF0800h)

```
Bit   Expl.
0     Disable 32K+256K WRAM (0=Normal, 1=Disable) (when off: empty/prefetch)
1-3   Unknown          (Read/Write-able)
4     Unknown          (Always zero, not used or write only)
5     Enable 256K WRAM (0=Disable, 1=Normal) (when off: mirror of 32K WRAM)
```

```
  6-23  Unknown          (Always zero, not used or write only)
  24-27 Wait Control WRAM 256K (0-14 = 15..1 Waitstates, 15=Lockup)
  28-31 Unknown          (Read/Write-able)
```
The default value 0Dh in Bits 24-27 selects 2 waitstates for 256K WRAM (ie. 3/3/6 cycles 8/16/32bit accesses). The fastest possible setting would be 0Eh (1 waitstate, 2/2/4 cycles for 8/16/32bit), that works on GBA and GBA SP only, the GBA Micro locks up with that setting (it's on-chip RAM is too slow, and works only with 2 or more waitstates).

Note: One cycle equals approx. 59.59ns (ie. 16.78MHz clock).

GBA GamePak Prefetch

GamePak Prefetch can be enabled in WAITCNT register. When prefetch buffer is enabled, the GBA attempts to read opcodes from Game Pak ROM during periods when the CPU is not using the bus (if any). Memory access is then performed with 0 Waits if the CPU requests data which is already stored in the buffer. The prefetch buffer stores up to eight 16bit values.

GamePak ROM Opcodes
The prefetch feature works only with <opcodes> fetched from GamePak ROM. Opcodes executed in RAM or BIOS are not affected by the prefetch feature (even if that opcodes read <data> from GamePak ROM).

Prefetch Enable
For GamePak ROM opcodes, prefetch may occur in two situations:
```
  1) opcodes with internal cycles (I) which do not change R15, shift/rotate
     register-by-register, load opcodes (ldr,ldm,pop,swp), multiply opcodes
  2) opcodes that load/store memory (ldr,str,ldm,stm,etc.)
```

Prefetch Disable Bug
When Prefetch is disabled, the Prefetch Disable Bug will occur for all
```
  "Opcodes in GamePak ROM with Internal Cycles which do not change R15"
```
for those opcodes, the bug changes the opcode fetch time from 1S to 1N.
Note: Affected opcodes (with I cycles) are: Shift/rotate register-by-register opcodes, multiply opcodes, and load opcodes (ldr,ldm,pop,swp).

GBA Cartridges

ROM
GBA Cartridge Header
GBA Cartridge ROM

Backup Media
Aside from ROM, cartridges may also include one of the following backup medias, used to store game positions, highscore tables, options, or other data.
GBA Cart Backup IDs
GBA Cart Backup SRAM/FRAM
GBA Cart Backup EEPROM
GBA Cart Backup Flash ROM
GBA Cart Backup DACS

Add-Ons
GBA Cart I/O Port (GPIO)
GBA Cart Real-Time Clock (RTC)

GBA Cart Solar Sensor
GBA Cart Tilt Sensor
GBA Cart Gyro Sensor
GBA Cart Rumble
GBA Cart e-Reader
GBA Cart Unknown Devices
GBA Cart Protections

Other Accessoires
GBA Flashcards
GBA Cheat Devices

GBA Cartridge Header

The first 192 bytes at 8000000h-80000BFh in ROM are used as cartridge header. The same header is also used for Multiboot images at 2000000h-20000BFh (plus some additional multiboot entries at 20000C0h and up).

Header Overview

```
  Address Bytes Expl.
  000h    4     ROM Entry Point  (32bit ARM branch opcode, eg. "B rom_start")
  004h    156   Nintendo Logo    (compressed bitmap, required!)
  0A0h    12    Game Title       (uppercase ascii, max 12 characters)
  0ACh    4     Game Code        (uppercase ascii, 4 characters)
  0B0h    2     Maker Code       (uppercase ascii, 2 characters)
  0B2h    1     Fixed value      (must be 96h, required!)
  0B3h    1     Main unit code   (00h for current GBA models)
  0B4h    1     Device type      (usually 00h) (bit7=DACS/debug related)
  0B5h    7     Reserved Area    (should be zero filled)
  0BCh    1     Software version (usually 00h)
  0BDh    1     Complement check (header checksum, required!)
  0BEh    2     Reserved Area    (should be zero filled)
  --- Additional Multiboot Header Entries ---
  0C0h    4     RAM Entry Point  (32bit ARM branch opcode, eg. "B ram_start")
  0C4h    1     Boot mode        (init as 00h - BIOS overwrites this value!)
  0C5h    1     Slave ID Number  (init as 00h - BIOS overwrites this value!)
  0C6h    26    Not used         (seems to be unused)
  0E0h    4     JOYBUS Entry Pt. (32bit ARM branch opcode, eg. "B joy_start")
```

Note: With all entry points, the CPU is initially set into system mode.

000h - Entry Point, 4 Bytes
Space for a single 32bit ARM opcode that redirects to the actual startaddress of the cartridge, this should be usually a "B <start>" instruction.
Note: This entry is ignored by Multiboot slave GBAs (in fact, the entry is then overwritten and redirected to a separate Multiboot Entry Point, as described below).

004h..09Fh - Nintendo Logo, 156 Bytes
Contains the Nintendo logo which is displayed during the boot procedure. Cartridge won't work if this data is missing or modified.
In detail: This area contains Huffman compression data (but excluding the compression header which is hardcoded in the BIOS, so that it'd be probably not possible to hack the GBA by producing de-compression buffer overflows).
A copy of the compression data is stored in the BIOS, the GBA will compare this data and lock-up itself if the BIOS data isn't exactly the same as in the cartridge (or multiboot header). The only exception are the two entries below which are allowed to have variable settings in some bits.

09Ch Bit 2,7 - Debugging Enable

This is part of the above Nintendo Logo area, and must be commonly set to 21h, however, Bit 2 and Bit 7 may be set to other values.
When both bits are set (ie. A5h), the FIQ/Undefined Instruction handler in the BIOS becomes unlocked, the handler then forwards these exceptions to the user handler in cartridge ROM (entry point defined in 80000B4h, see below).
Other bit combinations currently do not seem to have special functions.

09Eh Bit 0,1 - Cartridge Key Number MSBs
This is part of the above Nintendo Logo area, and must be commonly set to F8h, however, Bit 0-1 may be set to other values.
During startup, the BIOS performs some dummy-reads from a stream of pre-defined addresses, even though these reads seem to be meaningless, they might be intended to unlock a read-protection inside of commercial cartridge. There are 16 pre-defined address streams - selected by a 4bit key number - of which the upper two bits are gained from 800009Eh Bit 0-1, and the lower two bits from a checksum across header bytes 09Dh..0B7h (bytewise XORed, divided by 40h).

0A0h - Game Title, Uppercase Ascii, max 12 characters
Space for the game title, padded with 00h (if less than 12 chars).

0ACh - Game Code, Uppercase Ascii, 4 characters
This is the same code as the AGB-UTTD code which is printed on the package and sticker on (commercial) cartridges (excluding the leading "AGB-" part).

```
  U  Unique Code          (usually "A" or "B" or special meaning)
  TT Short Title          (eg. "PM" for Pac Man)
  D  Destination/Language (usually "J" or "E" or "P" or specific language)
```

The first character (U) is usually "A" or "B", in detail:

```
  A  Normal game; Older titles (mainly 2001..2003)
  B  Normal game; Newer titles (2003..)
  C  Normal game; Not used yet, but might be used for even newer titles
  F  Famicom/Classic NES Series (software emulated NES games)
  K  Yoshi and Koro Koro Puzzle (acceleration sensor)
  P  e-Reader (dot-code scanner)
  R  Warioware Twisted (cartridge with rumble and z-axis gyro sensor)
  U  Boktai 1 and 2 (cartridge with RTC and solar sensor)
  V  Drill Dozer (cartridge with rumble)
```

The second/third characters (TT) are:

```
  Usually an abbreviation of the game title (eg. "PM" for "Pac Man") (unless
  that gamecode was already used for another game, then TT is just random)
```

The fourth character (D) indicates Destination/Language:

```
  J  Japan             P  Europe/Elsewhere   F  French          S  Spanish
  E  USA/English       D  German             I  Italian
```

0B0h - Maker code, Uppercase Ascii, 2 characters
Identifies the (commercial) developer. For example, "01"=Nintendo.

0B2h - Fixed value, 1 Byte
Must be 96h.

0B3h - Main unit code, 1 Byte
Identifies the required hardware. Should be 00h for current GBA models.

0B4h - Device type, 1 Byte
Normally, this entry should be zero. With Nintendo's hardware debugger Bit 7 identifies the debugging handlers entry point and size of DACS (Debugging And Communication System) memory: Bit7=0: 9FFC000h/8MBIT DACS, Bit7=1: 9FE2000h/1MBIT DACS. The debugging handler can be enabled in 800009Ch (see above), normal cartridges do not have any memory (nor any mirrors) at these addresses though.

0B5h - Reserved Area, 7 Bytes
Reserved, zero filled.

0BCh - Software version number
Version number of the game. Usually zero.

0BDh - Complement check, 1 Byte
Header checksum, cartridge won't work if incorrect. Calculate as such:
chk=0:for i=0A0h to 0BCh:chk=chk-[i]:next:chk=(chk-19h) and 0FFh

0BEh - Reserved Area, 2 Bytes
Reserved, zero filled.

Below required for Multiboot/slave programs only. For Multiboot, the above 192 bytes are required to be transferred as header-block (loaded to 2000000h-20000BFh), and some additional header-information must be located at the beginning of the actual program/data-block (loaded to 20000C0h and up). This extended header consists of Multiboot Entry point(s) which must be set up correctly, and of two reserved bytes which are overwritten by the boot procedure:

0C0h - Normal/Multiplay mode Entry Point
This entry is used only if the GBA has been booted by using Normal or Multiplay transfer mode (but not by Joybus mode).
Typically deposit a ARM-32bit "B <start>" branch opcode at this location, which is pointing to your actual initialization procedure.

0C4h (BYTE) - Boot mode
The slave GBA download procedure overwrites this byte by a value which is indicating the used multiboot transfer mode.

Value	Expl.
01h	Joybus mode
02h	Normal mode
03h	Multiplay mode

Typically set this byte to zero by inserting DCB 00h in your source.
Be sure that your uploaded program does not contain important program code or data at this location, or at the ID-byte location below.

0C5h (BYTE) - Slave ID Number
If the GBA has been booted in Normal or Multiplay mode, this byte becomes overwritten by the slave ID number of the local GBA (that'd be always 01h for normal mode).

Value	Expl.
01h	Slave #1
02h	Slave #2
03h	Slave #3

Typically set this byte to zero by inserting DCB 00h in your source.
When booted in Joybus mode, the value is NOT changed and remains the same as uploaded from the master GBA.

0C6h..0DFh - Not used
Appears to be unused.

0E0h - Joybus mode Entry Point
If the GBA has been booted by using Joybus transfer mode, then the entry point is located at this address rather than at 20000C0h. Either put your initialization procedure directly at this address, or redirect to the actual boot procedure by depositing a "B <start>" opcode here (either one using 32bit ARM code). Or, if you are not intending to support joybus mode (which is probably rarely used), ignore this entry.

GBA Cartridge ROM

ROM Size
The games F-ZERO and Super Mario Advance use ROMs of 4 MBytes each. Zelda uses 8 MBytes. Not sure if other sizes are manufactured.

ROM Waitstates
The GBA starts the cartridge with 4,2 waitstates (N,S) and prefetch disabled. The program may change these settings by writing to WAITCNT, the games F-ZERO and Super Mario Advance use 3,1 waitstates (N,S) each, with prefetch enabled.
Third-party flashcards are reportedly running unstable with these settings. Also, prefetch and shorter waitstates are allowing to read more data and opcodes from ROM is less time, the downside is that it increases the power consumption.

ROM Chip
Because of how 24bit addresses are squeezed through the Gampak bus, the cartridge must include a circuit that latches the lower 16 address bits on non-sequential access, and that increments these bits on sequential access. Nintendo includes this circuit directly in the ROM chip.
Also, the ROM must have 16bit data bus (or a circuit which converts two 8bit data units into one 16bit unit - by not exceeding the waitstate timings).

GBA Cart Backup IDs

Nintendo didn't include a backup-type entry in the ROM header, however, the required type can be detected by ID strings in the ROM-image. Nintendo's tools are automatically inserting these strings (as part of their library headers). When using other tools, you may insert ID strings by hand.

ID Strings
The ID string must be located at a word-aligned memory location, the string length should be a multiple of 4 bytes (padded with zero's).

```
EEPROM_Vnnn    EEPROM 512 bytes or 8 Kbytes (4Kbit or 64Kbit)
SRAM_Vnnn      SRAM 32 Kbytes (256Kbit)
FLASH_Vnnn     FLASH 64 Kbytes (512Kbit) (ID used in older files)
FLASH512_Vnnn  FLASH 64 Kbytes (512Kbit) (ID used in newer files)
FLASH1M_Vnnn   FLASH 128 Kbytes (1Mbit)
```

For Nintendo's tools, "nnn" is a 3-digit library version number. When using other tools, best keep it set to "nnn" rather than inserting numeric digits.

Notes
No$gba does auto-detect most backup types, even without ID strings, except for 128K FLASH (without ID "FLASH1M_Vnnn", the FLASH size defaults to 64K). Ideally, for faster detection, the ID should be put into the first some bytes of the ROM-image (ie. somewhere right after the ROM header).

GBA Cart Backup SRAM/FRAM

SRAM - 32 KBytes (256Kbit) Lifetime: Depends on back-up battery
FRAM - 32 KBytes (256Kbit) Lifetime: 10,000,000,000 read/write per bit

Hyundai GM76V256CLLFW10 SRAM (Static RAM) (eg. F-Zero)

Fujitsu MB85R256 FRAM (Ferroelectric RAM) (eg. Warioware Twisted)

Addressing and Waitstates
SRAM/FRAM is mapped to E000000h-E007FFFh, it should be accessed with 8 waitstates (write a value of 3 into Bit0-1 of WAITCNT).

Databus Width
The SRAM/FRAM databus is restricted to 8 bits, it should be accessed by LDRB, LDRSB, and STRB opcodes only.

Reading and Writing
Reading from SRAM/FRAM should be performed by code executed in WRAM only (but not by code executed in ROM). There is no such restriction for writing.

Preventing Data Loss
The GBA SRAM/FRAM carts do not include a write-protect function (unlike older 8bit gameboy carts). This seems to be a problem and may cause data loss when a cartridge is removed or inserted while the GBA is still turned on. As far as I understand, this is not so much a hardware problem, but rather a software problem, ie. theoretically you could remove/insert the cartridge as many times as you want, but you should take care that your program does not crash (and write blindly into memory).

Recommended Workaround
Enable the Gamepak Interrupt (it'll most likely get triggered when removing the cartridge), and hang-up the GBA in an endless loop when your interrupt handler senses a Gamepak IRQ. For obvious reason, your interrupt handler should be located in WRAM, ie. not in the (removed) ROM cartridge. The handler should process Gamepak IRQs at highest priority. Periods during which interrupts are disabled should be kept as short as possible, if necessary allow nested interrupts.

When to use the above Workaround
A program that relies wholly on code and data in WRAM, and that does not crash even when ROM is removed, may keep operating without having to use the above mechanism.
Do NOT use the workaround for programs that run without a cartridge inserted (ie. single gamepak/multiboot slaves), or for programs that use Gamepak IRQ/DMA for other purposes.
All other programs should use it. It'd be eventually a good idea to include it even in programs that do not use SRAM/FRAM themselves (eg. otherwise removing a SRAM/FRAM-less cartridge may lock up the GBA, and may cause it to destroy backup data when inserting a SRAM/FRAM cartridge).

SRAM vs FRAM
FRAM (Ferroelectric RAM) is a newer technology, used in newer GBA carts, unlike SRAM (Static RAM), it doesn't require a battery to hold the data. At software side, it is accessed exactly like SRAM, ie. unlike EEPROM/FLASH, it doesn't require any Write/Erase commands/delays.

Note
In SRAM/FRAM cartridges, the /REQ pin (Pin 31 of Gamepak bus) should be a little bit shorter as than the other pins; when removing the cartridge, this causes the gamepak IRQ signal to get triggered before the other pins are disconnected.

GBA Cart Backup EEPROM

9853 - EEPROM 512 Bytes (0200h) (4Kbit) (eg. used by Super Mario Advance)
9854 - EEPROM 8 KBytes (2000h) (64Kbit) (eg. used by Boktai)
Lifetime: 100,000 writes per address

Addressing and Waitstates
The eeprom is connected to Bit0 of the data bus, and to the upper 1 bit (or upper 17 bits in case of large 32MB ROM) of the cartridge ROM address bus, communication with the chip takes place serially.
The eeprom must be used with 8 waitstates (set WAITCNT=X3XXh; 8,8 clks in WS2 area), the eeprom can be then addressed at DFFFF00h..DFFFFFFh.
Respectively, with eeprom, ROM is restricted to 8000000h-9FFFeFFh (max. 1FFFF00h bytes = 32MB minus 256 bytes). On carts with 16MB or smaller ROM, eeprom can be alternately accessed anywhere at D000000h-DFFFFFFh.

Data and Address Width
Data can be read from (or written to) the EEPROM in units of 64bits (8 bytes). Writing automatically erases the old 64bits of data. Addressing works in units of 64bits respectively, that is, for 512 Bytes EEPROMS: an address range of 0-3Fh, 6bit bus width; and for 8KByte EEPROMs: a range of 0-3FFh, 14bit bus width (only the lower 10 address bits are used, upper 4 bits should be zero).

Set Address (For Reading)
Prepare the following bitstream in memory:
```
  2 bits "11" (Read Request)
  n bits eeprom address (MSB first, 6 or 14 bits, depending on EEPROM)
  1 bit "0"
```
Then transfer the stream to eeprom by using DMA.

Read Data
Read a stream of 68 bits from EEPROM by using DMA,
then decipher the received data as follows:
```
  4 bits - ignore these
 64 bits - data (conventionally MSB first)
```

Write Data to Address
Prepare the following bitstream in memory, then transfer the stream to eeprom by using DMA, it'll take ca. 108368 clock cycles (ca. 6.5ms) until the old data is erased and new data is programmed.
```
  2 bits "10" (Write Request)
  n bits eeprom address (MSB first, 6 or 14 bits, depending on EEPROM)
 64 bits data (conventionally MSB first)
  1 bit "0"
```
After the DMA, keep reading from the chip, by normal LDRH [DFFFF00h], until Bit 0 of the returned data becomes "1" (Ready). To prevent your program from locking up in case of malfunction, generate a timeout if the chip does not reply after 10ms or longer.

Using DMA
Transferring a bitstream to/from the EEPROM by LDRH/STRH opcodes does not work, this might be because of timing problems, or because how the GBA squeezes non-sequential memory addresses through the external address/data bus.
For this reason, a buffer in memory must be used (that buffer would be typically allocated temporarily on stack, one halfword for each bit, bit1-15 of the halfwords are don't care, only bit0 is of interest).
The buffer must be transfered as a whole to/from EEPROM by using DMA3 (only DMA 3 is valid to read & write external memory), use 16bit transfer mode, both source and destination address incrementing (ie. DMA3CNT=80000000h+length).
DMA channels of higher priority should be disabled during the transfer (ie. H/V-Blank or Sound FIFO DMAs). And, of course any interrupts that might mess with DMA registers should be disabled.

Pin-Outs
The EEPROM chips are having only 8 pins, these are connected, Pin 1..8, to ROMCS, RD, WR, AD0, GND, GND, A23, VDD of the GamePak bus. Carts with 32MB ROM must have A7..A22 logically ANDed with A23.

Notes
There seems to be no autodection mechanism, so that a hardcoded bus width must be used.

GBA Cart Backup Flash ROM

64 KBytes - 512Kbits Flash ROM - Lifetime: 10,000 writes per sector
128 KBytes - 1Mbit Flash ROM - Lifetime: ??? writes per sector

Chip Identification (all device types)
```
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=90h  (enter ID mode)
  dev=[E000001h], man=[E000000h]                  (get device & manufacturer)
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=F0h  (terminate ID mode)
```
Used to detect the type (and presence) of FLASH chips. See Device Types below.

Reading Data Bytes (all device types)
```
  dat=[E00xxxxh]                                  (read byte from address xxxx)
```

Erase Entire Chip (all device types)
```
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=80h  (erase command)
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=10h  (erase entire chip)
  wait until [E000000h]=FFh (or timeout)
```
Erases all memory in chip, erased memory is FFh-filled.

Erase 4Kbyte Sector (all device types, except Atmel)
```
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=80h  (erase command)
  [E005555h]=AAh, [E002AAAh]=55h, [E00n000h]=30h  (erase sector n)
  wait until [E00n000h]=FFh (or timeout)
```
Erases memory at E00n000h..E00nFFFh, erased memory is FFh-filled.

Erase-and-Write 128 Bytes Sector (only Atmel devices)
```
  old=IME, IME=0                                  (disable interrupts)
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=A0h  (erase/write sector command)
  [E00xxxxh+00h..7Fh]=dat[00h..7Fh]               (write 128 bytes)
  IME=old                                         (restore old IME state)
  wait until [E00xxxxh+7Fh]=dat[7Fh] (or timeout)
```
Interrupts (and DMAs) should be disabled during command/write phase. Target address must be a multiple of 80h.

Write Single Data Byte (all device types, except Atmel)
```
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=A0h  (write byte command)
  [E00xxxxh]=dat                                  (write byte to address xxxx)
  wait until [E00xxxxh]=dat (or timeout)
```
The target memory location must have been previously erased.

Terminate Command after Timeout (only Macronix devices, ID=1CC2h)
```
  [E005555h]=F0h                              (force end of write/erase command)
```
Use if timeout occurred during "wait until" periods, for Macronix devices only.

Bank Switching (devices bigger than 64K only)
```
  [E005555h]=AAh, [E002AAAh]=55h, [E005555h]=B0h  (select bank command)
  [E000000h]=bnk                                  (write bank number 0..1)
```
Specifies 64K bank number for read/write/erase operations.
Required because gamepak flash/sram addressbus is limited to 16bit width.

Device Types

Nintendo puts different FLASH chips in commercial game cartridges. Developers should thus detect & support all chip types. For Atmel chips it'd be recommended to simulate 4K sectors by software, though reportedly Nintendo doesn't use Atmel chips in newer games anymore. Also mind that different timings should not disturb compatibility and performance.

ID	Name	Size	Sectors	AverageTimings	Timeouts/ms	Waits
D4BFh	SST	64K	16x4K	20us?,?,?	10, 40, 200	3,2
1CC2h	Macronix	64K	16x4K	?,?,?	10,2000,2000	8,3
1B32h	Panasonic	64K	16x4K	?,?,?	10, 500, 500	4,2
3D1Fh	Atmel	64K	512x128	?,?,?	...40.., 40	8,8
1362h	Sanyo	128K	?	?,?,?	? ? ? ?	
09C2h	Macronix	128K	?	?,?,?	? ? ? ?	

Identification Codes MSB=Device Type, LSB=Manufacturer.
Size in bytes, and numbers of sectors * sector size in bytes.
Average medium Write, Erase Sector, Erase Chips timings are unknown?
Timeouts in milliseconds for Write, Erase Sector, Erase Chips.
Waitstates for Writes, and Reads in clock cycles.

Accessing FLASH Memory
FLASH memory is located in the "SRAM" area at E000000h..E00FFFFh, which is restricted to 16bit address and 8bit data buswidths. Respectively, the memory can be accessed <only> by 8bit read/write LDRB/STRB opcodes.
Also, reading anything (data or status/busy information) can be done <only> by opcodes executed in WRAM (not from opcodes in ROM) (there's no such restriction for writing).

FLASH Waitstates
Use 8 clk waitstates for initial detection (WAITCNT Bits 0,1 both set). After detection of certain device types smaller wait values may be used for write/erase, and even smaller wait values for raw reading, see Device Types table.
In practice, games seem to use smaller values only for write/erase (even though those operations are slow anyways), whilst raw reads are always done at 8 clk waits (even though reads could actually benefit slightly from smaller wait values).

Verify Write/Erase and Retry
Even though device signalizes the completion of write/erase operations, it'd be recommended to read/confirm the content of the changed memory area by software. In practice, Nintendo's "erase-write-verify-retry" function typically repeats the operation up to three times in case of errors.
Also, for SST devices only, the "erase-write" and "erase-write-verify-retry" functions repeat the erase command up to 80 times, additionally followed by one further erase command if no retries were needed, otherwise followed by six further erase commands.

Note
FLASH (64Kbytes) is used by the game Sonic Advance, and possibly others.

GBA Cart Backup DACS

128 KBytes - 1Mbit DACS - Lifetime: 100,000 writes.
1024 KBytes - 8Mbit DACS - Lifetime: 100,000 writes.

DACS (Debugging And Communication System) is used in Nintendo's hardware debugger only, DACS is NOT used in normal game cartridges.

Parts of DACS memory is used to store the debugging exception handlers (entry point/size defined in cartridge header), the remaining memory could be used to store game positions or other data. The address space is the upper end of the 32MB ROM area, the memory can be read directly by the CPU, including for ability to execute

program code in this area.

GBA Cart I/O Port (GPIO)

4bit General Purpose I/O Port (GPIO) - contained in the ROM-chip

Used by Boktai for RTC and Solar Sensor:
GBA Cart Real-Time Clock (RTC)
GBA Cart Solar Sensor
And by Warioware Twisted for Rumble and Z-Axis Sensor:
GBA Cart Rumble
GBA Cart Gyro Sensor
Might be also used by other games for other purposes, such like other sensors, or SRAM bank switching, etc.

The I/O registers are mapped to a 6-byte region in the ROM-area at 80000C4h, the 6-byte region should be zero-filled in the ROM-image. In Boktai, the size of the zero-filled region is 0E0h bytes - that probably due to an incorrect definition (the additional bytes do not contain any extra ports, nor mirrors of the ports in the 6-byte region). Observe that ROM-bus writes are limited to 16bit/32bit access (STRB opcodes are ignored; that, only in DS mode?).

80000C4h - I/O Port Data (selectable W or R/W)
```
bit0-3  Data Bits 0..3 (0=Low, 1=High)
bit4-15 not used (0)
```

80000C6h - I/O Port Direction (for above Data Port) (selectable W or R/W)
```
bit0-3  Direction for Data Port Bits 0..3 (0=In, 1=Out)
bit4-15 not used (0)
```

80000C8h - I/O Port Control (selectable W or R/W)
```
bit0    Register 80000C4h..80000C8h Control (0=Write-Only, 1=Read/Write)
bit1-15 not used (0)
```
In write-only mode, reads return 00h (or possible other data, if the rom contains non-zero data at that location).

Connection Examples
```
GPIO       | Boktai   | Wario
Bit Pin    | RTC SOL  | GYR RBL
-----------+----------+---------
0   ROM.1  | SCK CLK  | RES -
1   ROM.2  | SIO RST  | CLK -
2   ROM.21 | CS  -    | DTA -
3   ROM.22 | -   FLG  | -   MOT
-----------+----------+---------
IRQ ROM.43 | IRQ -    | -   -
```

Aside from the I/O Port, the ROM-chip also includes an inverter (used for inverting the RTC /IRQ signal), and some sort of an (unused) address decoder output (which appears to be equal or related to A23 signal) (ie. reacting on ROM A23, or SRAM D7, which share the same pin on GBA slot).

GBA Cart Real-Time Clock (RTC)

S3511 - 8pin RTC with 3-wire serial bus (used in Boktai)

The RTC chip is (almost) the same as used in NDS consoles:

DS Real-Time Clock (RTC)
The chip is accessed via 4bit I/O port (only 3bits are used for RTC):
GBA Cart I/O Port (GPIO)

Comparision of RTC Registers

```
NDS_________GBA_________GBA/Params___
stat2       control     (1-byte)
datetime    datetime    (7-byte)
time        time        (3-byte)
stat1       force reset (0-byte)
clkadjust   force irq   (0-byte)
alarm1/int1 always FFh  (boktai contains code for writing 1-byte to it)
alarm2      always FFh  (unused)
free        always FFh  (unused)
```

Control Register

```
Bit Dir Expl.
0   -   Not used
1   R/W IRQ duty/hold related?
2   -   Not used
3   R/W Per Minute IRQ (30s duty)        (0=Disable, 1=Enable)
4   -   Not used
5   R/W Unknown?
6   R/W 12/24-hour Mode                  (0=12h, 1=24h) (usually 1)
7   R   Power-Off (auto cleared on read) (0=Normal, 1=Failure)
```

Setting after Battery-Shortcut is 82h. Setting after Force-Reset is 00h.
Unused bits seem to be always zero, but might be read-only or write-only?

Datetime and Time Registers
Same as NDS, except AM/PM flag moved from hour.bit6 (NDS) to hour.bit7 (GBA).

Force Reset/Irq Registers
Used to reset all RTC registers (all used registers become 00h, except day/month which become 01h), or to drag the IRQ output LOW for a short moment. These registers are strobed by ANY access to them, ie. by both writing to, as well as reading from these registers.

Pin-Outs / IRQ Signal
The package has identical pin-outs as in NDS, although it is slightly larger than the miniature chip in the DS. For whatever reason, the RTC's /IRQ output is passed through an inverter (contained in the ROM-chip), the inverted signal is then passed to the /IRQ pin on the cartridge slot. So, IRQ's will be triggered on the "wrong" edge - possible somehow in relation with detecting cartridge-removal IRQs?

GBA Cart Solar Sensor

Uses a Photo Diode as Solar Sensor (used in Boktai, allowing to defeat vampires when the cartridge is exposed to sunlight). The cartridge comes in transparent case, and it's slightly longer than normal carts, so the sensor reaches out of the cartridge slot. According to the manual, the sensor works only with sunlight, but actually it works with any strong light source (eg. a 100 Watt bulb at 1-2 centimeters distance). The sensor is accessed via 4bit I/O port (only 3bits used), which is contained in the ROM-chip.
GBA Cart I/O Port (GPIO)

A/D Conversion
The cartridge uses a self-made A/D converter, which is (eventually) better than measuring a capacitor charge-up time, and/or less expensive than a real ADC-chip:
It contains a 74LV4040 12bit binary counter (clocked by CPU via the I/O port), of which only the lower 8bit are

used, which are passed to a resistor ladder-type D/A converter, which is generating a linear increasing voltage, which is passed to a TLV272 voltage comparator, which is passing a signal to the I/O port when the counter voltage becomes greater than the sensor voltage.

Example Code

```
  strh  0001h,[80000c8h] ;-enable R/W mode
  strh  0007h,[80000c6h] ;-init I/O direction
  strh  0002h,[80000c4h] ;-reset counter to zero (high=reset) (I/O bit0)
  strh  0000h,[80000c4h] ;-clear reset (low=normal)
  mov   r0,0             ;-initial level
 @@lop:
  strh  0001h,[80000c4h] ;-clock high ;\increase counter      (I/O bit1)
  strh  0000h,[80000c4h] ;-clock low  ;/
  ldrh  r1,[80000c4h]    ;-read port                          (I/O bit3)
  tst   r1,08h           ;\
  addeq r0,1             ; loop until voltage match (exit with r0=00h..FFh),
  tsteq r0,100h          ; or until failure/timeout (exit with r0=100h)
  beq   @@lop            ;/
```

The results vary depending on the clock rate used. In above example, ensure that IRQs or DMAs do not interrupt the function. Alternately, use a super-slow clock rate (eg. like 666Hz used in Boktai) so that additional small IRQ/DMA delays have little effect on the overall timing. Results should be somewhat:

```
  E8h  total darkness (including daylight on rainy days)
  Dxh  close to a 100 Watt Bulb
  5xh  reaches max level in boktai's solar gauge
  00h  close to a tactical nuclear bomb dropped on your city
```

The exact values may change from cartridge to cartridge, so it'd be recommened to include a darkness calibration function, prompting the user to cover the sensor for a moment.

GBA Cart Tilt Sensor

Yoshi's Universal Gravitation / Yoshi Topsy Turvy (X/Y-Axis)
Koro Koro Puzzle (probably same as Yoshi, X/Y-Axis, too) (?)

Yoshi-Type (X/Y-Axis)

All of the registers are one byte wide, mapped into the top "half" of the SRAM memory range.

```
  E008000h (W) Write 55h to start sampling
  E008100h (W) Write AAh to start sampling
  E008200h (R) Lower 8 bits of X axis
  E008300h (R) Upper 4 bits of X axis, and Bit7: ADC Status (0=Busy, 1=Ready)
  E008400h (R) Lower 8 bits of Y axis
  E008500h (R) Upper 4 bits of Y axis
```

You must set SRAM wait control to 8 clocks to access it correctly.
You must also set the cartridge PHI terminal to 4 MHz to make it work.
Sampling routine (typically executed once a frame during VBlank):

```
  wait until [E008300h].Bit7=1 or until timeout ;wait ready
  x = ([E008300h] AND 0Fh)*100h + [E008200h]    ;get x
  y = ([E008500h] AND 0Fh)*100h + [E008400h]    ;get y
  [E008000h]=55h, [E008100h]=AAh                ;start next conversion
```

Example values (may vary on different carts and on temperature, etc):

```
  X ranged between 0x2AF to 0x477, center at 0x392.    Huh?
  Y ranged between 0x2C3 to 0x480, center at 0x3A0.    Huh?
```

Thanks to Flubba for Yoshi-Type information.
Unknown if the Yoshi-Type sensors are sensing rotation, or orientation, or motion, or something else? In case of rotation, rotation around X-axis would result in motion in Y-direction, so not too sure whether X and Y have which meaning?
Most probably, the sensors are measuring (both) static acceleration (gravity), and dynamic acceleration (eg. shaking the device left/right).

The X/Y values are likely to be mirrored depending on using a back-loading cartridge slot (original GBA), or front-loading cartridge slot (newer GBA SP, and NDS, and NDS-Lite).

GBA Cart Gyro Sensor

Warioware Twisted (Z-Axis Gyro Sensor, plus Rumble)

Wario-Type (Z-Axis)
Uses a single-axis sensor, which senses rotation around the Z-axis. The sensor is connected to an analogue-in, serial-out ADC chip, which is accessed via lower 3 bits of the GPIO,
GBA Cart I/O Port (GPIO)
The four I/O Lines are connected like so,

```
GPIO.Bit0 (W) Start Conversion
GPIO.Bit1 (W) Serial Clock
GPIO.Bit2 (R) Serial Data
GPIO.Bit3 (W) Used for Rumble (not gyro related)
```

There should be at least <three sequential 32bit ARM opcodes executed in WS0 region> between the STRH opcodes which toggle the CLK signal. Wario uses WAITCNT=45B7h (SRAM=8clks, WS0/WS1/WS2=3,1clks, Prefetch=On, PHI=Off).
The data stream consists of: 4 dummy bits (usually zero), followed by 12 data bits, followed by endless unused bits (usually zero).

```
read_gyro:
  mov  r1,8000000h      ;-cartridge base address
  mov  r0,01h           ;\enable R/W access
  strh r0,[r1,0c8h]     ;/
  mov  r0,0bh           ;\init direction (gpio2=input, others=output)
  strh r0,[r1,0c6h]     ;/
  ldrh r2,[r1,0c4h]     ;-get current state (for keeping gpio3=rumble)
  orr  r2,3                     ;\
  strh r2,[r1,0c4h] ;gpio0=1    ; start ADC conversion
  bic  r2,1                     ;
  strh r2,[r1,0c4h] ;gpio0=0    ;/
  mov  r0,00010000h ;stop-bit           ;\
  bic  r2,2                             ;
@@lop:                                  ;
  ldrh r3,[r1,0c4h] ;get gpio2=data     ; read 16 bits
  strh r2,[r1,0c4h] ;gpio1=0=clk=low    ; (4 dummy bits, plus 12 data bits)
  movs r3,r3,lsr 3  ;gpio2 to cy=data   ;
  adcs r0,r0,r0     ;merge data, cy=done;
  orr  r3,r2,2      ;set bit1 and delay ;
  strh r3,[r1,0c4h] ;gpio1=1=clk=high   ;
  bcc  @@lop                            ;/
  bic  r0,0f000h                ;-strip upper 4 dummy bits (isolate 12bit adc)
  bx   lr
```

Example values (may vary on different carts, battery charge, temperature, etc):

```
  354h  rotated in anti-clockwise direction (shock-speed)
  64Dh  rotated in anti-clockwise direction (normal fast)
  6A3h  rotated in anti-clockwise direction (slow)
  6C0h  no rotation                         (stopped)
  6DAh  rotation in clockwise direction     (slow)
  73Ah  rotation in clockwise direction     (normal fast)
  9E3h  rotation in clockwise direction     (shock-speed)
```

For detection, values 000h and FFFh would indicate that there's no sensor.
The Z-axis always points into same direction; no matter of frontloading or backloading cartridge slots.
Thanks to Momo Vampire for contributing a Wario cartridge.

X/Y/Z-Axes
X-Axis and Y-Axis are meant to be following the screens X and Y coordinates, so the Z-Axis would point into

the screens depth direction.

DSi Cameras
DSi consoles can mis-use the built-in cameras as Gyro sensor (as done by the System Flaw DSi game).

GBA Cart Rumble

Warioware Twisted (Rumble, plus Z-Axis Gyro Sensor)
Drill Dozer (Rumble only) <-- and ALSO supports Gameboy Player rumble?

GBA Rumble Carts are containing a small motor, which is causing some vibration when/while it is switched on (that, unlike DS Rumble, which must be repeatedly toggled on/off).

In Warioware Twisted, rumble is controlled via GPIO.Bit3 (Data 0=Low=Off, 1=High=On) (and Direction 1=Output), the other GPIO Bits are used for the gyro sensor.
GBA Cart I/O Port (GPIO)
Note: GPIO3 is connected to an external pulldown resistor (so the HighZ level gets dragged to Low=Off when direction is set to Input).

Unknown if Drill Dozer is controlled via GPIO.Bit3, too?

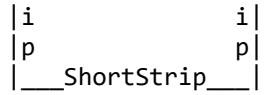
DS Rumble Pak
Additionally, there's a Rumble Pak for the NDS, which connects to the GBA slot, so it can be used also for GBA games (provided that the game doesn't require the GBA slot, eg. GBA multiboot games).
DS Cart Rumble Pak

Gamecube Rumble
Moreover, GBA games that are running on a Gameboy Player are having access to the Rumble function of Gamecube joypads.
GBA Gameboy Player

GBA Cart e-Reader

GBA Cart e-Reader Overview
GBA Cart e-Reader I/O Ports
GBA Cart e-Reader Dotcode Format
GBA Cart e-Reader Data Format
GBA Cart e-Reader Program Code
GBA Cart e-Reader API Functions
GBA Cart e-Reader VPK Decompression
GBA Cart e-Reader Error Correction
GBA Cart e-Reader File Formats

```
 _______________
|   ShortStrip  |
|L              L|
|o    Center    o|
|n    Region    n|
|g              g|
|  may contain  |
|S   pictures,  S|
|t instructions t|
|r     etc.     r|
```



GBA Cart e-Reader Overview

The e-Reader is a large GBA cartridge (about as big as the GBA console), with built-in dotcode scanning hardware. Dotcodes are tiny strips of black and white pixels printed on the edges of cardboard cards. The cards have to be pulled through a slot on the e-Reader, which is giving it a feeling like using a magnet card reader. The binary data on the dotcodes contains small games, either in native GBA code (ARM/THUMB), or in software emulated 8bit Z80 or NES/Famicom (6502) code.

The e-Reader Hardware
The hardware consists of regular 8MByte ROM and 128KByte FLASH chips, two link ports, a custom PGA chip, the camera module (with two red LEDs, used as light source), and some analogue components for generating the LED voltages, etc. The camera supports 402x302 pixels with 7bit monochrome color depth, but the PGA clips it to max 320 pixels per scanline with 1bit color depth.

Link Port Plug/Socket
The e-Reader's two link ports are simply interconnected with each other; without connection to the rest of the e-Reader hardware. These ports are used only on the original GBA (where the large e-Reader cartridge would be covering the GBA's link socket). When trying to insert the e-Reader into an original NDS (or GBA-Micro), then the e-Reader's link plug will hit against the case of the NDS, so it works only with some minor modification to the hardware. There's no such problem with GBA-SP and NDS-Lite.

Region/Version
There are 3 different e-Reader's: Japanese/Original, Japanese/Plus, and Non-Japanese. The Original version has only 64K FLASH, no Link Port, and reportedly supports only Z80 code, but no NES/GBA code. The Plus and Non-Japanese versions should be almost identical, except that they reject cards from the wrong region, and that the title strings aren't ASCII in Japan, the Plus version should be backwards compatible to the Original one.

The Problem
Nintendo's current programmers are definetly unable to squeeze a Pac-Man style game into less than 4MBytes. Their solution has been: MORE memory. That is, they've put a whopping 8MByte BIOS ROM into the e-Reader, which contains the User Interface, and software emulation for running some of their 20 years old 8bit NES and Game&Watch titles, which do fit on a few dotcode strips.

GBA Cart e-Reader I/O Ports

DF80000h Useless Register (R/W)

0	Output to PGA.Pin93 (which seems to be not connected to anything)
1-3	Unknown, read/write-able (not used by e-Reader BIOS)
4-15	Always zero (0)

DFA0000h Reset Register (R/W)

0	Always zero	(0)
1	Reset Something?	(0=Normal, 1=Reset)
2	Unknown, always set	(1)
3	Unknown, read/write-able	(not used by e-Reader BIOS)
4-7	Always zero	(0)
8	Unknown, read/write-able	(not used by e-Reader BIOS)
9-15	Always zero	(0)

DFC0000h..DFC0027h Scanline Data (R)
Scanline data (40 bytes, for 320 pixels, 1bit per pixel, 0=black, 1=white).
The first (leftmost) pixel is located in the LSB of the LAST byte.
Port E00FFB1h.Bit1 (and [4000202h].Bit13) indicates when a new scanline is present, the data should be then transferred to RAM via DMA3 (SAD=DFC0000h, DAD=buf+y*28h, CNT=80000014h; a slower non-DMA transfer method would result in missed scanlines). After the DMA, software must reset E00FFB1h.Bit1.
Note: The scanning resolution is 1000 DPI.

DFC0028h+(0..2Fh*2) Brightest Pixels of 8x6 Blocks (R)
```
  0-6  Max Brightness (00h..7Fh; 00h=All black, 7Fh=One or more white)
  7-15 Always zero
```
Can be used to adjust the Port E00FF80h..E00FFAFh settings.

DFC0088h Darkest Pixel of whole Image (R)
```
  0-7  Max Darkness   (00h..7Fh; 00h=One or more black, 7Fh=All white)
  8-15 Always zero
```
Can be used to adjust the Port E00FF80h..E00FFAFh settings.

E00FF80h..E00FFAFh Intensity Boundaries for 8x6 Blocks (R/W)
The 320x246 pixel camera input is split into 8x6 blocks (40x41 pixels each), with Block00h=Upper-right, Block07h=Upper-left, ..., Block27h=Lower-left. The boundary values for the separate blocks are used for 128-grayscale to 2-color conversion, probably done like "IF Pixel>Boundary THEN white ELSE black".
```
  0-6  Block Intensity Boundaries (0..7Fh; 7Fh=Whole block gets black)
  7    Always zero
```
The default boundary values are stored in FLASH memory, the values are typically ranging from 28h (outer edges) to 34h (center image), that in respect to the light source (the two LEDs are emitting more light to the center region).

E00FFB0h Control Register 0 (R/W)
```
  0    Serial Data       (Low/High)
  1    Serial Clock      (Low/High)
  2    Serial Direction  (0=Input, 1=Output)
  3    Led/Irq Enable    (0=Off, 1=On; Enable LED and Gamepak IRQ)
  4    Start Scan        (0=Off, 1=Start) (0-to-1 --> Resync line 0)
  5    Phi 16MHz Output  (0=Off, 1=On; Enable Clock for Camera, and for LED)
  6    Power 3V Enable   (0=Off, 1=On; Enable 3V Supply for Camera)
  7    Not used          (always 0) (sometimes 1) (Read only)
```

E00FFB1h Control Register 1 (R/W)
```
  0    Not used          (always 0)
  1    Scanline Flag     (1=Scanline Received, 0=Acknowledge)
  2-3  Not used          (always 0)
  4    Strange Bit       (0=Normal, 1=Force Resync/Line0 on certain interval?)
  5    LED Anode Voltage (0=3.0V, 1=5.1V; requires E00FFB0h.Bit3+5 to be set)
  6    Not used          (always 0)
  7    Input from PGA.Pin22, always high (not used by e-Reader) (Read Only)
```
Bit1 can be SET by hardware only, software can only RESET that bit, the Gamepak IRQ flag (Port 4000202h.Bit13) becomes set on 0-to-1 transitions.

E00FFB2h Light Source LED Kathode Duration (LSB) (R/W)
E00FFB3h Light Source LED Kathode Duration (MSB) (R/W)
Selects the LED Kathode=LOW Duration, aka the LED=ON Duration. That does act as pulse width modulated LED brightness selection (the camera seems to react slowly enough to view the light as being dimmed to medium, rather than seeing the actual light ON and OFF states). The PWM timer seems to be clocked at 8MHz. The hardware clips timer values 2000h..FFFFh to max 2000h (=1ms). Additionally, the e-Reader BIOS clips values to max 11B3h. Default setting is found in FLASH calibration data. A value of 0000h disables the LED.

Serial Port Registers (Camera Type 1) (DV488800) (calib_data[3Ch]=1)

All 16bit values are ordered MSB,LSB. All registers are whole 8bit Read/Write-able, except 00h,57h-5Ah (read only), and 53h-55h (2bit only).

```
  Port     Expl.               (e-Reader Setting)
  00h      Maybe Chip ID (12h) (not used by e-Reader BIOS) (Read Only)
  01h                          (05h)    ;-Bit0: 1=auto-repeat scanning?
  02h                          (0Eh)
  10h-11h  Vertical Scroll     (calib_data[30h]+7)
  12h-13h  Horizontal Scroll   (0030h)
  14h-15h  Vertical Size       (00F6h=246)
  16h-17h  Horizontal Size     (0140h=320)
  20h-21h  H-Blank Duration    (00C4h)
  22h-23h                      (0400h)  ;-Upper-Blanking in dot-clock units?
  25h                          (var)    ;-bit1: 0=enable [57h..5Ah] ?
  26h                          (var)    ;\maybe a 16bit value
  27h                          (var)    ;/
  28h                          (00h)
  30h      Brightness/contrast (calib_data[31h]+/-nn)
  31h-33h                      (014h,014h,014h)
  34h      Brightness/contrast (02h)
  50h-52h  8bit Read/Write     (not used by e-Reader BIOS)
  53h-55h  2bit Read/Write     (not used by e-Reader BIOS)
  56h      8bit Read/Write     (not used by e-Reader BIOS)
  57h-58h  16bit value, used to autodetect/adjust register[30h] (Read Only)
  59h-5Ah  16bit value, used to autodetect/adjust register[30h] (Read Only)
  80h-FFh  Mirrors of 00h..7Fh (not used by e-Reader BIOS)
```

All other ports are unused, writes to those ports are ignored, and reads are returning data mirrored from other ports; that is typically data from 2 or more ports, ORed together.

Serial Port Registers (Camera Type 2) (calib_data[3Ch]=2)

All 16bit values are using more conventional LSB,MSB ordering, and port numbers are arranged in a more reasonable way. The e-Reader BIOS doesn't support (or doesn't require) brightness adjustment for this camera module.

```
  Port     Expl.            (e-Reader Setting)
  00h                       (22h)
  01h                       (50h)
  02h-03h  Vertical Scroll  (calib_data[30h]+28h)
  04h-05h  Horizontal Scroll (001Eh)
  06h-07h  Vertical Size    (00F6h)    ;=246
  08h-09h  Horizontal Size  (0140h)    ;=320
  0Ah-0Ch                   (not used by e-Reader BIOS)
  0Dh                       (01h)
  0Eh-0Fh                   (01EAh)    ;=245*2
  10h-11h                   (00F5h)    ;=245
  12h-13h                   (20h,F0h)  ;maybe min/max values?
  14h-15h                   (31h,C0h)  ;maybe min/max values?
  16h                       (00h)
  17h-18h                   (77h,77h)
  19h-1Ch                   (30h,30h,30h,30h)
  1Dh-20h                   (80h,80h,80h,80h)
  21h-FFh                   (not used by e-Reader BIOS)
```

This appears to be a Micron (aka Aptina) camera (resembling the DSi cameras).

My own e-Reader uses a Type 1 camera module. Not sure if Nintendo has ever manufactured any e-Readers with Type 2 cameras?

Calibration Data in FLASH Memory (Bank 0, Sector 0Dh)

```
  E00D000 14h  ID String ('Card-E Reader 2001',0,0)
  E00D014 2    Sector Checksum (NOT(x+x/10000h); x=sum of all other halfwords)
```

Begin of actual data (40h bytes)

```
  E00D016 8x6  [00h] Intensity Boundaries for 8x6 blocks ;see E00FF80h..AFh
```

```
  E00D046 1    [30h] Vertical scroll (0..36h)  ;see type1.reg10h/type2.reg02h
  E00D047 1    [31h] Brightness or contrast    ;see type1.reg30h
  E00D048 2    [32h] LED Duration              ;see E00FFB2h..B3h
  E00D04A 2    [34h] Not used?   (0000h)
  E00D04C 2    [36h] Signed value, related to adjusting the 8x6 blocks
  E00D04E 4    [38h] Not used?   (00000077h)
  E00D052 4    [3Ch] Camera Type (0=none,1=DV488800,2=Whatever?)
```
Remaining bytes in this Sector...
```
  E00D056 FAAh Not used (zerofilled) (included in above checksum)
```

Flowchart for Overall Camera Access

ereader_scan_camera:
```
 call ereader_power_on
 call ereader_initialize
 for z=1 to number_of_frames
  for y=0 to 245
   Wait until E00FFB1h.Bit1 gets set by hardware (can be handled by IRQ)
   Copy 14h halfwords from DFC0000h to buf+y*28h via DMA3
   Reset E00FFB1h.Bit1 by software
  next y
  ;(could now check DFC0028h..DFC0086h/DFC0088h for adjusting E00FF00h..2Fh)
  ;(could now show image on screen, that may require to stop/pause scanning)
 next z
 call ereader_power_off
 Ret
```
ereader_power_on:
```
 [4000204h]=5803h   ;Init waitstates, and enable Phi 16MHz
 [DFA0000h].Bit1=1
 Wait(10ms)
 [E00FFB0h]=40h     ;Enable Power3V and reset other bits
 [DFA0000h].Bit1=0
 [E00FFB1h]=20h     ;Enable Power5V and reset other bits
 Wait(40ms)
 [E00FFB1h].Bit4=0  ;...should be already 0 ?
 [E00FFB0h]=40h+27h ;Phi16MHz=On, SioDtaClkDir=HighHighOut
 Ret
```
ereader_power_off:
```
 [E00FFB0h]=04h    ;Power3V=Off, Disable Everything, SioDtaClkDir=LowLowOut
 [DFA0000h].Bit1=0 ;...should be already 0
 [E00FFB1h].Bit5=0 ;Power5V=Off
 Ret
```
ereader_initialize:
```
 IF calib_data[3Ch] AND 03h = 1 THEN init_camera_type1
 [E00FFB0h].Bit4=1 ;ScanStart
 IF calib_data[3Ch] AND 03h = 2 THEN init_camera_type2
 Copy calib_data[00h..2Fh] to [E00FF80h+00h..2Fh]  ;Intensity Boundaries
 Copy calib_data[32h..33h] to [E00FFB2h+00h..01h]  ;LED Duration LSB,MSB
 [E00FFB0h].Bit3=1                                 ;LedIrqOn
 Ret
```
init_camera_type1:
```
 x=MIN(0,calib_data[31h]-0Bh)
 Set Sio Registers (as shown for Camera Type 1, except below values...)
 Set Sio Registers [30h]=x [25h]=04h, [26h]=58h, [27h]=6Ch
 ;(could now detect/adjust <x> based on Sio Registers [57h..5Ah])
 Set Sio Registers [30h]=x [25h]=06h, [26h]=E8h, [27h]=6Ch
 Ret
```
init_camera_type2:
```
 Wait(0.5ms)
 Set Sio Registers (as shown for Camera Type 2)
 Ret
```

Accessing Serial Registers via E00FFB0h
```
      Begin   Write(A) Write(B) Read(C) Read(D) End     Idle    PwrOff
```

```
 Dir ooooooo ooooooo  ooooooo  iiiiiii iiiiiii ooooooo ooooooo ooooooo
 Dta ---____ AAAAAAA  BBBBBBB  xxxxxCx xxxxxDx ______- ------- _______
 Clk ------_ ___---_  ___---_  ___---_ ___---_ ___---- ------- _______
```

Flowchart for accessing Serial Registers via E00FFB0h (looks like I2C bus)

```
Delay:
 Wait circa 2.5us, Ret
SioBegin:
 SioDta=1, SioDir=Out, SioClk=1, Delay, SioDta=0, Delay, SioClk=0, Ret
SioEnd:
 SioDta=0, SioDir=Out, Delay, SioClk=1, Delay, SioDta=1, Ret
SioRead1bit:   ;out: databit
 SioDir=In, Delay, SioClk=1, Delay, databit=SioDta, SioClk=0, Ret
SioWrite1bit:  ;in: databit
 SioDta=databit, SioDir=Out, Delay, SioClk=1, Delay, SioClk=0, Ret
SioReadByte:   ;in: endflag - out: data
 for i=7 to 0, data.bit<i>=SioRead1bit, next i, SioWrite1bit(endflag), Ret
SioWriteByte:  ;in: data - out: errorflag
 for i=7 to 0, Delay(huh/why?), SioWrite1bit(data.bit<i>), next i
 errorflag=SioRead1bit, SioDir=Out(huh/why?), Ret
SioWriteRegisters:  ;in: index, len, buffer
 SioBegin
 SioWriteByte(22h)        ;command (set_index) (and write_data)
 SioWriteByte(index)      ;index
 for i=0 to len-1
  SioWriteByte(buffer[i]) ;write data (and auto-increment index)
 next
 SioEnd
 ret
SioReadRegisters:   ;in: index, len - out: buffer
 SioBegin
 SioWriteByte(22h)        ;command (set_index) (without any write_data here)
 SioWriteByte(index)      ;index
 SioBegin
 SioWriteByte(23h)        ;command (read_data) (using above index)
 for i=0 to len-1
  if i=len-1 then endflag=1 else endflag=0
  buffer[i]=SioReadByte(endflag)  ;read data (and auto-increment index)
 next
 SioEnd
 Ret
```

Caution: Accessing the SIO registers appears highly unstable, and seems to require error handling with retries. Not sure what is causing that problem, possibly the registers cannot be accessed during camera-data-scans...?

WAITCNT

The e-Reader BIOS uses WAITCNT [4000204h]=5803h when accessing the PGA, that is, gamepak 16.78MHz phi output (bit11-12=3), 8 waits for SRAM region (bit0-1=3), gamepak prefetch enabled (bit14=1), also sets WS0 to 4,2 waits (bit2-4=0), and sets WS2 to odd 4,8 waits (bit8-10=0). The WS2 (probably WS0 too) settings are nonsense, and should work with faster timings (the e-Reader can be accessed in NDS mode, which doesn't support that slow timings).

e-Reader Memory and I/O Map (with all used/unused/mirrored regions)

```
 C000000h-C7FFFFFh  ROM (8MB)
 C800000h-DF7FFFFh  Open Bus
 DF80000h-DF80001h  Useless Register (R/W)
 DF80002h-DF9FFFFh  Mirrors of DF80000h-DF80001h
 DFA0000h-DFA0001h  Reset Register (R/W)
 DFA0002h-DFBFFFFh  Mirrors of DFA0000h-DFA0001h
 DFC0000h-DFC0027h  Scanline Data (320 Pixels) (R)
 DFC0028h-DFC0087h  Brightest Pixels of 8x6 Blocks (R)
 DFC0088h           Darkest Pixel of whole Image (R)
 DFC0089h-DFC00FFh  Always zero
```

```
DFC0100h-DFDFFFFh  Mirrors of DFC0000h-DFC00FFh
DFE0000h-DFFFFFFh  Open Bus
E000000h-E00CFFFh  FLASH Bank 0 - Data
E00D000h-E00DFFFh  FLASH Bank 0 - Calibration Data
E00E000h-E00EFFFh  FLASH Bank 0 - Copy of Calibration Data
E00F000h-E00FF7Fh  FLASH Bank 0 - Unused region
E000000h-E00EFFFh  FLASH Bank 1 - Data
E00F000h-E00FF7Fh  FLASH Bank 1 - Unused region
E00FF80h-E00FFAFh  Intensity Boundaries for 8x6 Blocks (R/W)
E00FFB0h           Control Register 0 (R/W)
E00FFB1h           Control Register 1 (R/W)
E00FFB2h-E00FFB3h  LED Duration (16bit) (R/W)
E00FFB4h-E00FFBFh  Always zero
E00FFC0h-E00FFFFh  Mirror of E00FF80h-E00FFBFh
```

Mind that WS2 should be accessed by LDRH/STRH, and SRAM region by LDRB/STRB.
Additionally about 32 serial bus registers are contained in the camera module.

Camera Module Notes

The Type 1 initial setting on power-on is 402x302 pixels, the e-Reader uses only 320x246 pixels. The full vertical resolution could be probably used without problems. Port DFC0000h-DFC0027h are restricted to 320 pixels, so larger horizontal resolutions could be probably obtained only by changing the horizontal scroll offset on each 2nd scan.
The camera output is 128 grayscales (via parallel 7bit databus), but the PGA converts it to 2 colors (1bit depth). For still images, it might be possible to get 4 grayshades via 3 scans with different block intensity boundary settings.
No idea if the camera supports serial commands other than 22h and 23h. Namely, it <would> be a quite obvious and basic feature to allow to receive the bitmap via the 2-wire serial bus (alternately to the 7bit databus), if supported, it'd allow to get 7bit images, bypassing 1bit PGA conversion.
When used as actual camera (by cutting an opening in the case), the main problem is the 1bit color depth, which allows only black and white schemes, when/if solving that problem, focusing might be also a problem.

Either the camera or the PGA seem to have a problem on white-to-black transitions in vertical direction, the upper some black pixels are sorts of getting striped or dithered. For example, scanning the large sync marks appears as:

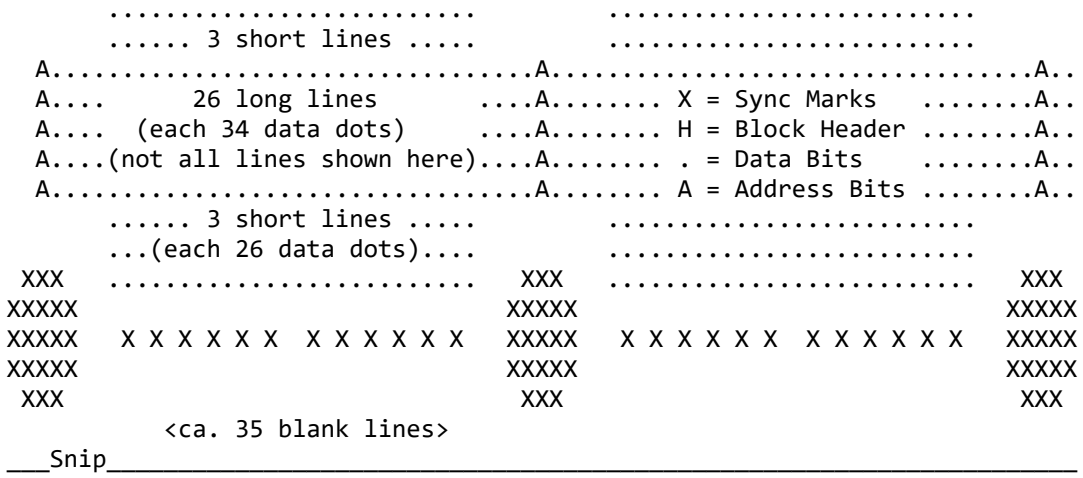
```
Actual Shape    Scanned Shape
   XXXXX            X  X
  XXXXXXX           X  X X
 XXXXXXXXX        X X  X XX
 XXXXXXXXX        X X  X XX
  XXXXXXX          XXXXXXX
   XXXXX            XXXXX
```

That appears only on large black shapes (the smaller data dots look better). Probably the image is scanned from bottom upwards (and the camera senses only the initial transition at the bottom, and then looses track of what it is doing).

GBA Cart e-Reader Dotcode Format

Resolution is 342.39 DPI (almost 10 blocks per inch).
Resolution is 134.8 dots/cm (almost 4 blocks per centimeter).
The width and height of each block, and the spacing to the bottom edge of the card is ca. 1/10 inch, or ca. 4 millimeters.

```
 XXX            BLOCK 1             XXX            BLOCK 2             XXX
XXXXX                              XXXXX                              XXXXX
XXXXX   X X X X X X  X X X X X X  XXXXX   X X X X X X  X X X X X X  XXXXX
XXXXX                              XXXXX                              XXXXX
 XXX   HHHHHHHHHHHHHHHHHHHHH.....   XXX   HHHHHHHHHHHHHHHHHHHHH.....   XXX
```



Address Columns

Each Column consists of 26 dots. From top to bottom: 1 black dot, 8 blank dots, 16 address dots (MSB topmost), and 1 blank dot. The 16bit address values can be calculated as:

```
addr[0] = 03FFh
for i = 1 to 53
  addr[i] = addr[i-1] xor ((i and (-i)) * 769h)
  if (i and 07h)=0 then addr[i] = addr[i] xor (769h)
  if (i and 0Fh)=0 then addr[i] = addr[i] xor (769h*2)
  if (i and 1Fh)=0 then addr[i] = addr[i] xor (769h*4) xor (769h)
next i
```

Short strips use addr[1..19], long strips use addr[25..53], left to right.

Block Header

The 18h-byte Block Header is taken from the 1st two bytes (20 dots) of the 1st 0Ch blocks (and is then repeated in the 1st two bytes of further blocks).

```
00h      Unknown              (00h)
01h      Dotcode type         (02h=Short, 03h=Long)
02h      Unknown              (00h)
03h      Address of 1st Block (01h=Short, 19h=Long)
04h      Total Fragment Size  (40h) ;64 bytes per fragment, of which,
                                    ;48 bytes are actual data, the remaining
05h      Error-Info Size      (10h) ;16 bytes are error-info
06h      Unknown              (00h)
07h      Interleave Value     (1Ch=Short, 2Ch=Long)
08h..17h 16 bytes Reed-solomon error correction info for Block Header
```

Data 4-Bit to 5-bit Conversion

In the Block Header (HHHHH), and Data Region (.....), each 4bit are expanded to 5bit, so one byte occupies 10 dots, and each block (1040 data dots) contains 104 bytes.

```
4bit  00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh
5bit  00h 01h 02h 12h 04h 05h 06h 16h 08h 09h 0Ah 14h 0Ch 0Dh 11h 10h
```

That formatting ensures that there are no more than two continous black dots (in horizontal direction), neither inside of a 5bit value, nor between two 5bit values, however, the address bars are violating that rule, and up to 5 continous black dots can appear at the (..A..) block boundaries.

Data Order

Data starts with the upper bit of the 5bit value for the upper 4bit of the first byte, which is located at the leftmost dot of the upper line of the leftmost block, it does then extend towards rightmost dot of that block, and does then continue in the next line, until reaching the bottom of the block, and does then continue in the next block. The 1st two bytes of each block contain a portion of the Block Header, the remaining 102 bytes in each block contain data.

Data Size

A long strip consists of 28 blocks (28*104 = 2912 bytes), a short strip of 18 blocks (18*104 = 1872 bytes). Of which, less than 75% can be actually used for program code, the remaining data contains error correction info, and various headers. See Data Format for more info.

Interleaved Fragments
The Interleave Value (I) specifies the number of fragments, and does also specify the step to the next byte inside of a fragment; except that, at the block boundaries (every 104 bytes), the step is 2 bigger (for skipping the next two Block Header bytes).

```
RAW Offset  Content
000h..001h  1st 2 bytes of RAW Header
002h        1st byte of 1st fragment
003h        1st byte of 2nd fragment
...         ...
002h+I-1    1st byte of last fragment
002h+I      2nd byte of 1st fragment
003h+I      2nd byte of 2nd fragment
...         ...
002h+I*2-1  2nd byte of last fragment
...         ...
```

Each fragment consists of 48 actual data bytes, followed by 16 error correction bytes, followed by 0..2 unused bytes (since I*40h doesn't exactly match num_blocks*102).

GBA Cart e-Reader Data Format

Data Strip Format
The size of the data region is I*48 bytes (I=Interleave Value, see Dotcode Format), the first 48-byte fragment contains the Data Header, the remaining (I-1) fragments are Data Fragments (which contain title(s), and VPK compressed program code).

First Strip

```
Data Header  (48 bytes)
Main-Title   (17 bytes, or 33 bytes)
Sub-Title(s) (3+18 bytes, or 33 bytes) (for each strip) (optional)
VPK Size     (2 byte value, total length of VPK Data in ALL strips)
NULL Value   (4 bytes, contained ONLY in 1st strip of GBA strips)
VPK Data     (length as defined in VPK Size entry, see above)
```

Further Strip(s)

```
Data Header  (48 bytes)
Main-Title   (17 bytes, or 33 bytes)
Sub-Title(s) (3+18 bytes, or 33 bytes) (for each strip) (optional)
VPK Data     (continued from previous strip)
```

Data Header (30h bytes) (1st fragment)

```
00h-01h  Fixed         (00h,30h)
02h      Fixed         (01h)     ;01h="Do not calculate Global Checksum" ?
03h      Primary Type  (see below)
04h-05h  Fixed         (00h,01h) (don't care)
06h-07h  Strip Size    (0510h=Short, 0810h=Long Strip) ((I-1)*30h) (MSB,LSB)
08h-0Bh  Fixed         (00h,00h,10h,12h)
0Ch-0Dh  Region/Type   (see below)
0Eh      Strip Type    (02h=Short Strip, 01h=Long Strip) (don't care)
0Fh      Fixed         (00h) (don't care)
10h-11h  Unknown       (whatever) (don't care)
12h      Fixed         (10h)     ;10h="Do calculate Data Checksum" ?
13h-14h  Data Checksum (see below) (MSB,LSB)
15h-19h  Fixed         (19h,00h,00h,00h,08h)
1Ah-21h  ID String     ('NINTENDO')
```

```
  22h-25h  Fixed         (00h,22h,00h,09h)
  26h-29h  Size Info     (see below)
  2Ah-2Dh  Flags         (see below)
  2Eh      Header Checksum (entries [0Ch-0Dh,10h-11h,26h-2Dh] XORed together)
  2Fh      Global Checksum (see below)
```
Primary Type [03h] is 8bit,
```
  0      Card Type (upper bit) (see below)
  1      Unknown (usually opposite of Bit0) (don't care)
  2-7    Unknown (usually zero)
```
Region/Type [0Ch..0Dh] is 16bit,
```
  0-3    Unknown (don't care)
  4-7    Card Type (lower bits) (see below)
  8-11   Region/Version (0=Japan/Original, 1=Non-japan, 2=Japan/Plus)
  12-15  Unknown (don't care)
```
Size Info [26h-29h] is 32bit,
```
  0      Unknown            (don't care)
  1-4    Strip Number       (01h..Number of strips)
  5-8    Number of Strips   (01h..0Ch) (01h..08h for Japan/Original version)
  9-23   Size of all Strips (excluding Headers and Main/Sub-Titles)
         (same as "VPK Size", but also including the 2-byte "VPK Size" value,
         plus the 4-byte NULL value; if it is present)
  24-31  Fixed              (02h) (don't care)
```
Flags [2Ah-2Dh] is 32bit,
```
  0      Permission to save (0=Start Immediately, 1=Prompt for FLASH Saving)
  1      Sub-Title Flag     (0=Yes, 1=None)    (Japan/Original: always 0=Yes)
  2      Application Type   (0=GBA/Z80, 1=NES) (Japan/Original: always 0=Z80)
  3-31   Zero (0) (don't care)
```
Data Checksum [13h-14h] is the complement (NOT) of the sum of all halfwords in all Data Fragments, however, it's all done in reversed byte order: checksum is calculated with halfwords that are read in MSB,LSB order, and the resulting checksum is stored in MSB,LSB order in the Header Fragment.
Global Checksum [2Fh] is the complement (NOT) of the sum of the first 2Fh bytes in the Data Header plus the sum of all Data Fragment checksums; the Data Fragment checksums are all 30h bytes in a fragment XORed with each other.

Titles (3+N bytes, or N bytes)
Titles can be 33 bytes for both Main and Sub (Format 0Eh), or Main=17 bytes and Sub=3+18 bytes (Formats 02h..05h). In the 3+N bytes form, the first 3 bytes (24bit) are are used to display "stats" information in form of "HP: h1 ID: i1-i2-i3", defined as:
```
  Bit    Expl.
  0-3    h1, values 1..15 shown as "10..150", value 0 is not displayed
  4-6    i3, values 0..7 shown as "A..G,#"
  7-13   i2, values 0..98 shown as "01..99" values 99..127 as "A0..C8"
  14-18  i1, values 0..31 shown as "A..Z,-,_,{HP},.,{ID?},:"
  19-22  Unknown
  23     Disable stats (0=Show as "HP: h1 ID: i1-i2-i3", 1=Don't show it)
```
The N bytes portion contains the actual title, which must be terminated by 00h (so the max length is N-1 characters, if it is shorter than N-1, then the unused bytes are padded by further 00h's). The character set is normal ASCII for non-Japan (see Region/Version entry in header), and 2-byte SHIFT-JIS for Japanese long-titles (=max 16 2-byte chars) with values as so:
```
  00h          --> end-byte
  81h,40h      --> SPC
  81h,43h..97h --> punctuation marks
  82h,4Fh..58h --> "0..9"
  82h,60h..79h --> "A..Z"
  82h,81h..9Ah --> "a..z"
```
And 1-byte chars for Japanese short-titles,
```
  00     = end-byte
  01     = spc
  02..0B = 0..9
  0C..AF = japanese
  B0..B4 = dash, male, female, comma, round-dot
```

```
B5..C0 = !"%&~?/+-:.'
C1..DA = A..Z
DB..DF = unused (blank)
E0..E5 = japanese
E6..FF = a..z
N/A    = #$()*;<=>@[\]^_`{|}
```
Additionally to the Main-Title, optional Sub-Titles for each strip can be included (see Sub-Title Flag in header). If enabled, then ALL strip titles are included in each strip (allowing to show a preview of which strips have/haven't been scanned yet).
The e-Reader can display maximum of 8 sub-titles, if the data consists of more than 8 strips, then sub-titles aren't displayed (so it'd be waste of space to include them in the dotcodes).
The Main Title gets clipped to 128 pixels width (that are, circa 22 characters), and, the e-Reader BIOS acts confused on multi-strip games with Main Titles longer than 26 characters (so the full 33 bytes may be used only in Japan; with 16bit charset).
If the title is empty (00h-filled), and there is only one card in the application, then the application is started immediately. That, without allowing the user to save it in FLASH memory.
Caution: Although shorter Titles do save memory, they do act unpleasant: the text "(C) P-Letter" will be displayed at the bottom of the loading screen.
On Japanese/Original, 8bit sub-titles can be up to 18 characters (without any end-byte) (or less when stats are enabled, due to limited screen width).

Card Types (Primary Type.Bit0 and Region/Type.Bit12-15)
```
00h..01h  Blank Screen (?)
02h..03h  Dotcode Application with 17byte-title, with stats, load music A
04h..05h  Dotcode Application with 17byte-title, with stats, load music B
06h..07h  P-Letter Attacks
08h..09h  Construction Escape
0Ah..0Bh  Construction Action
0Ch..0Dh  Construction Melody Box
0Eh       Dotcode Application with 33byte-title, without stats, load music A
0Fh       Game specific cards
10h..1Dh  P-Letter Viewer
1Eh..1Fh  Same as 0Eh and 0Fh (see above)
```
The 'Application' types are meant to be executable GBA/Z80/NES programs.

GBA Cart e-Reader Program Code

The GBA/Z80/NES program code is stored in the VPK compressed area.
NES-type is indicated by header [2Ah].Bit2, GBA-type is indicated by the NULL value inserted between VPK Size and VPK Data, otherwise Z80-type is used.

GBA Format
Load Address and Entrypoint are at 2000000h (in ARM state). The 32bit word at 2000008h is eventually destroyed by the e-Reader. Namely,
```
IF e-Reader is Non-Japanese,
AND [2000008h] is outside of range of 2000000h..20000E3h,
AND only if booted from camera (not when booted from FLASH?),
THEN [2000008h]=[2000008h]-0001610Ch ELSE [2000008h] kept intact
```
Existing multiboot-able GBA binaries can be converted to e-Reader format by,
```
Store "B 20000C0h" at 2000000h   ;redirect to RAM-entrypoint
Zerofill 2000004h..20000BFh      ;erase header (for better compression rate)
Store 01h,01h at 20000C4h        ;indicate RAM boot
```
The GBA code has full access to the GBA hardware, and may additionally use whatever API functions contained in the e-Reader BIOS. With the incoming LR register value, "mov r0,N, bx lr" returns to the e-Reader BIOS (with N being 0=Restart, or 2=To_Menu). No idea if it's necessary to preserve portions of RAM when returning to the e-Reader BIOS?

Caution: Unlike for normal GBA cartridges/multiboot files, the hardware is left uninitialized when booting dotcodes (among others: sound DMA is active, and brightness is set to zero), use "mov r0,0feh, swi 010000h" to get the normal settings.

NES Format

Emulates a NES (Nintendo Entertainment System) console (aka Family Computer).
The visible 240x224 pixel NES/NTSC screen resolution is resampled to 240x160 to match the smaller vertical resolution of the GBA hardware. So, writing e-Reader games in NES format will result in blurred screen output.
The screen/sound/joypad is accessed via emulated NES I/O ports, program code is running on an emulated 6502 8bit CPU, for more info on the NES hardware, see no$nes debugger specifications, or

```
http://problemkaputt.de/everynes.htm
```

The e-Reader's NES emulator supports only 16K PRG ROM, followed by 8K VROM. The emulation accuracy is very low, barely working with some of Nintendo's own NES titles; running the no$nes diagnostics program on it has successfully failed on ALL hardware tests ;-)
The load address for the 16K PRG-ROM is C000h, the 16bit NMI vector at [FFFAh] is encrypted like so:

```
for i=17h to 0
 for j=07h to 0, nmi = nmi shr 1, if carry then nmi = nmi xor 8646h, next j
 nmi = nmi xor (byte[dmca_data+i] shl 8)
next i
dmca_data: db 0,0,'DMCA NINTENDO E-READER'
```

The 16bit reset vector at [FFFCh] contains:

```
Bit0-14  Lower bits of Entrypoint (0..7FFFh = Address 8000h..FFFFh)
Bit15    Nametable Mode (0=Vertical Mirroring, 1=Horizontal Mirroring)
```

reportedly,

```
 (NES limitations, 1 16K program rom + 1-2 8K CHR rom, mapper 0 and 1)
 ines mapper 1 would be MMC1, rather than CNROM (ines mapper 3)?
 but, there are more or less NONE games that have 16K PRG ROM + 16K VROM?
```

The L+R Button key-combination allows to reset the NES, however, there seems to be no way to return to the e-Reader BIOS.

Z80/8080 Format

The e-Reader doesn't support the following Z80 opcodes:

```
CB [Prefix]     E0 RET PO   E2 JP PO,nn   E4 CALL PO,nn   27 DAA    76 HALT
ED [Prefix]     E8 RET PE   EA JP PE,nn   EC CALL PE,nn   D3 OUT (n),A
DD [IX Prefix]  F3 DI       08 EX AF,AF'  F4 CALL P,nn    DB IN A,(n)
FD [IY Prefix]  FB EI       D9 EXX        FC CALL M,nn    xx RST 00h..38h
```

That is leaving not more than six supported Z80 opcodes (DJNZ, JR, JR c/nc/z/nz), everything else are 8080 opcodes. Custom opcodes are:

```
76 WAIT A frames, D3 WAIT n frames, and C7/CF RST 0/8 used for API calls.
```

The load address and entrypoint are at 0100h in the emulated Z80 address space. The Z80 doesn't have direct access to the GBA hardware, instead video/sound/joypad are accessed via API functions, invoked via RST 0 and RST 8 opcodes, followed by an 8bit data byte, and with parameters in the Z80 CPU registers. For example, "ld a,02h, rst 8, db 00h" does return to the e-Reader BIOS.
The Z80/8080 emulation is incredibly inefficient, written in HLL code, developed by somebody whom knew nothing about emulation nor about ARM nor about Z80/8080 processors.

Running GBA-code on Japanese/Original e-Reader

Original e-Reader supports Z80 code only, but can be tweaked to run GBA-code:

```
retry:
 ld bc,data // ld hl,00c8h      ;src/dst
lop:
 ld a,[bc] // inc bc // ld e,a  ;lsb
 ld a,[bc] // inc bc // ld d,a  ;msb
 dw 0bcfh ;aka rst 8 // db 0bh  ;[4000000h+hl]=de (DMA registers)
 inc hl // inc  hl // ld a,l
 cp a,0dch // jr nz,lop
mod1 equ $+1
 dw 37cfh ;aka rst 8 // db 37h  ;bx 3E700F0h
```

```
;below executed only on jap/plus... on jap/plus, above 37cfh is hl=[400010Ch]
 ld a,3Ah // ld [mod1],a                  ;bx 3E700F0h (3Ah instead 37h)
 ld hl,1 // ld [mod2],hl // ld [mod3],hl  ;base (0200010Ch instead 0201610Ch)
 jr retry
data:
mod2 equ $+1
 dd loader         ;40000C8h dma2sad (loader)            ;\
 dd 030000F0h      ;40000CCh dma2dad (mirrored 3E700F0h) ; relocate loader
 dd 8000000ah      ;40000D0h dma2cnt (copy 0Ah x 16bit)  ;/
mod3 equ $+1
 dd main           ;40000D4h dma3sad (main)              ;\prepare main reloc
 dd 02000000h      ;40000D8h dma3dad (2000000h)          ;/dma3cnt see loader
 .align 2          ;alignment for 16bit-halfword
org $+201600ch     ;jap/plus: adjusted to org $+200000ch
loader:
 mov r0,80000000h  ;(dma3cnt, copy 10000h x 16bit)
 mov r1,04000000h  ;i/o base
 strb r1,[r1,208h] ;ime=0 (better disable ime before moving ram)
 str r0,[r1,0DCh]  ;dma3cnt (relocate to 2000000h)
 mov r15,2000000h  ;start relocated code at 2000000h in ARM state
main:
 ;...insert/append whatever ARM code here...
 end
```

GBA Cart e-Reader API Functions

Z80 Interface (Special Opcodes)

```
db 76h       ;Wait8bit A
db D3h,xxh   ;Wait8bit xxh
db C7h,xxh   ;RST0_xxh
db CFh,xxh   ;RST8_xxh
ld r,[00xxh]       ;get system values (addresses differ on jap/ori)
ld r,[00C2h..C3h]  ;GetKeyStateSticky (jap/ori: 9F02h..9F03h)
ld r,[00C4h..C5h]  ;GetKeyStateRaw    (jap/ori: 9F04h..9F05h)
ld r,[00C0h..C1h]  ;see Exit and ExitRestart
ld r,[00D0h..D3h]  ;see Mul16bit
```

For jap/ori, 9Fxxh isn't forwards compatible with jap/plus, so it'd be better to check joypad via IoRead.

GBA Interface

```
bx [30075FCh] ;ApiVector ;in: r0=func_no,r1,r2,r3,[sp+0],[sp+4],[sp+8]=params
bx lr         ;Exit      ;in: r0 (0=Restart, 2=To_Menu)
```

Wait8bit/Wait16bit

The various Wait opcodes and functions are waiting as many frames as specified. Many API functions have no effect until the next Wait occurs.

Z80 RST0_xxh Functions / GBA Functions 02xxh

```
RST0_00h FadeIn, A speed, number of frames (0..x)
RST0_01h FadeOut
RST0_02h BlinkWhite
RST0_03h  (?)
RST0_04h  (?) blend_func_unk1
RST0_05h  (?)
RST0_06h  (?)
RST0_07h  (?)
RST0_08h  (?)
RST0_09h  (?) _020264CC_check
RST0_0Ah  (?) _020264CC_free
RST0_0Bh N/A (bx 0)
RST0_0Ch N/A (bx 0)
```

```
RST0_0Dh N/A (bx 0)
RST0_0Eh N/A (bx 0)
RST0_0Fh N/A (bx 0)
RST0_10h LoadSystemBackground, A number of background (1..101), E bg# (0..3)
RST0_11h SetBackgroundOffset, A=bg# (0..3), DE=X, BC=Y
RST0_12h SetBackgroundAutoScroll
RST0_13h SetBackgroundMirrorToggle
RST0_14h  (?)
RST0_15h  (?)
RST0_16h  (?) write_000000FF_to_02029494_
RST0_17h  (?)
RST0_18h  (?)
RST0_19h SetBackgroundMode, A=mode (0..2)
RST0_1Ah  (?)
RST0_1Bh  (?)
RST0_1Ch  (?)
RST0_1Dh  (?)
RST0_1Eh  (?)
RST0_1Fh  (?)
RST0_20h LayerShow
RST0_21h LayerHide
RST0_22h  (?)
RST0_23h  (?)
RST0_24h ... [20264DCh+A*20h+1Ah]=DE, [20264DCh+A*20h+1Ch]=BC
RST0_25h  (?)
RST0_26h  (?)
RST0_27h  (?)
RST0_28h  (?)
RST0_29h  (?)
RST0_2Ah  (?)
RST0_2Bh  (?)
RST0_2Ch  (?)
RST0_2Dh LoadCustomBackground, A bg# (0..3), DE pointer to struct_background,
         max. tile data size = 3000h bytes, max. map data size = 1000h bytes
RST0_2Eh GBA: N/A - Z80: (?)
RST0_2Fh  (?)
RST0_30h CreateSystemSprite, - -   (what "- -" ???)
RST0_31h SpriteFree, HL sprite handle
RST0_32h SetSpritePos, HL=sprite handle, DE=X, BC=Y
RST0_33h  (?) sprite_unk2
RST0_34h SpriteFrameNext
RST0_35h SpriteFramePrev
RST0_36h SetSpriteFrame, HL=sprite handle, E=frame number (0..x)
RST0_37h  (?) sprite_unk3
RST0_38h  (?) sprite_unk4
RST0_39h SetSpriteAutoMove, HL=sprite handle, DE=X, BC=Y
RST0_3Ah  (?) sprite_unk5
RST0_3Bh  (?) sprite_unk6
RST0_3Ch SpriteAutoAnimate
RST0_3Dh  (?) sprite_unk7
RST0_3Eh SpriteAutoRotateUntilAngle
RST0_3Fh SpriteAutoRotateByAngle
RST0_40h SpriteAutoRotateByTime
RST0_41h  (?) sprite_unk8
RST0_42h SetSpriteAutoMoveHorizontal
RST0_43h SetSpriteAutoMoveVertical
RST0_44h  (?) sprite_unk9
RST0_45h SpriteDrawOnBackground
RST0_46h SpriteShow, HL=sprite handle
RST0_47h SpriteHide, HL=sprite handle
RST0_48h SpriteMirrorToggle
RST0_49h  (?) sprite_unk10
RST0_4Ah  (?) sprite_unk11
RST0_4Bh  (?) sprite_unk12
RST0_4Ch GetSpritePos
```

```
RST0_4Dh CreateCustomSprite
RST0_4Eh  (?)
RST0_4Fh  (?) sprite_unk14
RST0_50h  (?) sprite_unk15
RST0_51h  (?) sprite_unk16
RST0_52h  (?) sprite_unk17
RST0_53h  (?) sprite_unk18
RST0_54h  (?)
RST0_55h  (?) sprite_unk20
RST0_56h  (?)
RST0_57h SpriteMove
RST0_58h  (?) sprite_unk22
RST0_59h  (?) sprite_unk23
RST0_5Ah  (?) sprite_unk24
RST0_5Bh SpriteAutoScaleUntilSize, C=speed (higher value is slower),
         HL=sprite handle, DE=size (0100h = normal size,
         lower value = larger, higher value = smaller)
RST0_5Ch SpriteAutoScaleBySize
RST0_5Dh SpriteAutoScaleWidthUntilSize
RST0_5Eh SpriteAutoScaleHeightBySize
RST0_5Fh  (?)
RST0_60h  (?)
RST0_61h  (?)
RST0_62h  (?)
RST0_63h  (?)
RST0_64h hl=[[2024D28h+a*4]+12h]
RST0_65h  (?) sprite_unk25
RST0_66h SetSpriteVisible, HL=sprite handle, E=(0=not visible, 1=visible)
RST0_67h  (?) sprite_unk26
RST0_68h  (?) set_sprite_unk27
RST0_69h  (?) get_sprite_unk27
RST0_6Ah  (?)
RST0_6Bh  (?)
RST0_6Ch  (?)
RST0_6Dh  (?)
RST0_6Eh hl=[hl+000Ah]  ;r0=[r1+0Ah]
RST0_6Fh  (?)
RST0_70h  (?)
RST0_71h  (?)
RST0_72h  (?)
RST0_73h  (?)
RST0_74h  (?)
RST0_75h  (?)
RST0_76h  (?)
RST0_77h  (?)
RST0_78h  (?)
RST0_79h  (?)
RST0_7Ah  (?)
RST0_7Bh  (?)
RST0_7Ch  (?) _0202FD2C_unk12
RST0_7Dh Wait16bit ;HL=num_frames (16bit variant of Wait8bit opcode/function)
RST0_7Eh SetBackgroundPalette, HL=src_addr, DE=offset, C=num_colors (1..x)
RST0_7Fh GetBackgroundPalette(a,b,c)
RST0_80h SetSpritePalette, HL=src_addr, DE=offset, C=num_colors (1..x)
RST0_81h GetSpritePalette(a,b,c)
RST0_82h ClearPalette
RST0_83h  (?) _0202FD2C_unk11
RST0_84h  (?)
RST0_85h  (?)
RST0_86h  (?)
RST0_87h  (?) _0202FD2C_unk8
RST0_88h  (?) _0202FD2C_unk7
RST0_89h  (?)
RST0_8Ah  (?) _0202FD2C_unk6
RST0_8Bh  (?) _0202FD2C_unk5
```

```
RST0_8Ch GBA: N/A - Z80: (?)
RST0_8Dh GBA: N/A - Z80: (?)
RST0_8Eh  (?)
RST0_8Fh WindowHide
RST0_90h CreateRegion, H=bg# (0..3), L=palbank# (0..15),
         D,E,B,C=x1,y1,cx,cy (in tiles), return: n/a (no$note: n/a ???)
RST0_91h SetRegionColor
RST0_92h ClearRegion
RST0_93h SetPixel
RST0_94h GetPixel
RST0_95h DrawLine
RST0_96h DrawRect
RST0_97h  (?) _0202FD2C_unk4
RST0_98h SetTextColor, A=region handle, D=color foreground (0..15),
         E=color background (0..15)
RST0_99h DrawText, A=region handle, BC=pointer to text, D=X, E=Y
         (non-japan uses ASCII text, but japanese e-reader's use STH ELSE?)
RST0_9Ah SetTextSize
RST0_9Bh  (?) RegionUnk7
RST0_9Ch  (?) _0202FD2C_unk3
RST0_9Dh  (?) _0202FD2C_unk2
RST0_9Eh  (?) _0202FD2C_unk1
RST0_9Fh Z80: (?) - GBA: SetBackgroundModeRaw
RST0_A0h  (?)
RST0_A1h  (?)
RST0_A2h  (?) RegionUnk6
RST0_A3h GBA: N/A - Z80: (?)
RST0_A4h GBA: N/A - Z80: (?)
RST0_A5h  (?)
RST0_A6h  (?)
RST0_A7h  (?)
RST0_A8h  (?)
RST0_A9h  (?)
RST0_AAh  (?)
RST0_ABh  (?)
RST0_ACh  (?)
RST0_ADh  (?) RegionUnk5
RST0_AEh [202FD2Ch+122h]=A
RST0_AFh [202FD2Ch+123h]=A
RST0_B0h [202FD2Ch+124h]=A
RST0_B1h  (?)
RST0_B2h  (?)
RST0_B3h GBA: N/A - Z80: Sqrt   ;hl=sqrt(hl)
RST0_B4h GBA: N/A - Z80: ArcTan ;hl=ArcTan2(hl,de)
RST0_B5h Sine                   ;hl=sin(a)*de
RST0_B6h Cosine                 ;hl=cos(a)*de
RST0_B7h  (?)
RST0_B8h  (?)
RST0_B9h N/A (bx 0)
RST0_BAh N/A (bx 0)
RST0_BBh N/A (bx 0)
RST0_BCh N/A (bx 0)
RST0_BDh N/A (bx 0)
RST0_BEh N/A (bx 0)
RST0_BFh N/A (bx 0)
```
Below Non-Japan and Japan/Plus only (not Japan/Ori)
```
RST0_C0h GetTextWidth(a,b)
RST0_C1h GetTextWidthEx(a,b,c)
RST0_C2h  (?)
RST0_C3h Z80: N/A (bx 0) - GBA: (?)
RST0_C4h  (?)
RST0_C5h  (?)
RST0_C6h  (?)
RST0_C7h  (?)
RST0_C8h  (?)
```

```
RST0_C9h  (?)
RST0_CAh  (?)
RST0_CBh  (?)
RST0_CCh  (?)
RST0_CDh N/A (bx lr)
RST0_CEh ;same as RST0_3Bh, but with 16bit mask
RST0_CFh ;same as RST0_3Eh, but with 16bit de
RST0_D0h ;same as RST0_3Fh, but with 16bit de
RST0_D1h ;same as RST0_5Bh, but with 16bit de
RST0_D2h ;same as RST0_5Ch, but with 16bit de
RST0_D3h ;same as RST0_5Dh, but with 16bit de
RST0_D4h ;same as RST0_5Eh, but with 16bit de
RST0_D5h  (?)
RST0_D6h  (?)
RST0_D7h ;[202FD2Ch+125h]=A
RST0_D8h  (?)
RST0_D9h  (?)
RST0_DAh  (?)
RST0_DBh ;A=[3003E51h]
RST0_DCh ;[3004658h]=01h
RST0_DDh DecompressVPKorNonVPK
RST0_DEh FlashWriteSectorSingle(a,b)
RST0_DFh FlashReadSectorSingle(a,b)
RST0_E0h SoftReset
RST0_E1h GetCartridgeHeader     ;[hl+0..BFh]=[8000000h..80000BFh]
RST0_E2h GBA: N/A - Z80: bx hl  ;in: hl=addr, af,bc,de,sp=param, out: a
RST0_E3h Z80: N/A (bx 0) - GBA: (?)
RST0_E4h  (?)
RST0_E5h  (?)
RST0_E6h  (?)
RST0_E7h  (?)
RST0_E8h  (?)
RST0_E9h ;[2029498h]=0000h
RST0_EAh Z80: N/A (bx 0) - GBA: InitMemory(a)
RST0_EBh  (?) BL_irq_sio_dma3
RST0_ECh ;hl = [3003E30h]*100h + [3003E34h]
RST0_EDh FlashWriteSectorMulti(a,b,c)
RST0_EEh FlashReadPart(a,b,c)
RST0_EFh ;A=((-([2029416h] xor 1)) OR (+([2029416h] xor 1))) SHR 31
RST0_F0h  (?) _unk1
RST0_F1h RandomInit     ;in: hl=random_seed
RST0_F2h                         (?)
Below Japan/Plus only
RST0_F3h  (?)
RST0_F4h  (?)
RST0_F5h  (?)
RST0_F6h  (?)
RST0_F7h GBA: N/A - Z80: (?)
Below is undefined/garbage (values as so in Z80 mode)
Jap/Ori: RST0_C0h      N/A (bx 0)
Jap/Ori: RST0_C1h..FFh Overlaps RST8 jump list
Non-Jap: RST0_F3h..FFh Overlaps RST8 jump list
Jap/Pls: RST0_F8h..FFh Overlaps RST8 jump list
```

Z80 RST8_xxh Functions / GBA Functions 01xxh

```
RST8_00h GBA: N/A - Z80: Exit       ;[00C0h]=a ;(1=restart, 2=exit)
RST8_01h GBA: N/A - Z80: Mul8bit    ;hl=a*e
RST8_02h GBA: N/A - Z80: Mul16bit   ;hl=hl*de, s32[00D0h]=hl*de
RST8_03h Div                        ;hl=hl/de
RST8_04h DivRem                     ;hl=hl mod de
RST8_05h PlaySystemSound            ;in: hl=sound_number
RST8_06h  (?) sound_unk1
RST8_07h Random8bit                 ;a=random(0..FFh)
RST8_08h SetSoundVolume
```

```
RST8_09h BcdTime                    ;[de+0..5]=hhmmss(hl*bc)
RST8_0Ah BcdNumber                  ;[de+0..4]=BCD(hl), [de+5]=00h
RST8_0Bh IoWrite                    ;[4000000h+hl]=de
RST8_0Ch IoRead                     ;de=[4000000h+hl]
RST8_0Dh GBA: N/A - Z80:  (?)
RST8_0Eh GBA: N/A - Z80:  (?)
RST8_0Fh GBA: N/A - Z80:  (?)
RST8_10h GBA: N/A - Z80:  (?)
RST8_11h DivSigned                  ;hl=hl/de, signed
RST8_12h RandomMax                  ;a=random(0..a-1)
RST8_13h SetSoundSpeed
RST8_14h  hl=[202FD20h]=[2024CACh]
RST8_15h  hl=[2024CACh]-[202FD20h]
RST8_16h SoundPause
RST8_17h SoundResume
RST8_18h PlaySystemSoundEx
RST8_19h IsSoundPlaying
RST8_1Ah  (?)
RST8_1Bh  (?)
RST8_1Ch  (?)
RST8_1Dh GetExitCount               ;a=[2032D34h]
RST8_1Eh Permille                   ;hl=de*1000/hl
RST8_1Fh GBA: N/A - Z80: ExitRestart;[2032D38h]=a, [00C0h]=0001h  ;a=?
RST8_20h GBA: N/A - Z80: WaitJoypad ;wait until joypad<>0, set hl=joypad
RST8_21h GBA: N/A - Z80:  (?)
RST8_22h  (?) _sound_unk7
RST8_23h  (?) _sound_unk8
RST8_24h  (?) _sound_unk9
RST8_25h  (?) _sound_unk10
RST8_26h Mosaic     ;bg<n>cnt.bit6=a.bit<n>, [400004Ch]=de
RST8_27h  (?)
RST8_28h  (?)
RST8_29h  (?)
RST8_2Ah  (?) get_8bit_from_2030110h
RST8_2Bh  (?)
RST8_2Ch  (?) get_16bit_from_2030112h ;jap/ori: hl=[20077B2h]
RST8_2Dh  (?) get_16bit_from_2030114h ;jap/ori: hl=[20077B4h]
RST8_2Eh  (?)
RST8_2Fh PlayCustomSound(a,b)
Below not for Japanese/Original
(the renumbered functions can be theoretically used on japanese/original)
(but, doing so would blow forwards compatibility with japanese/plus)
RST8_30h (ori: none)      GBA: N/A - Z80: (?)
RST8_31h (ori: none)      PlayCustomSoundEx(a,b,c)
RST8_32h (ori: RST8_30h)  BrightnessHalf   ;[4000050h]=00FFh,[4000054h]=0008h
RST8_33h (ori: RST8_31h)  BrightnessNormal ;[4000050h]=0000h
RST8_34h (ori: RST8_32h)  N/A (bx lr)
RST8_35h (ori: RST8_33h)   (?)
RST8_36h (ori: RST8_34h)  ResetTimer ;[400010Ch]=00000000h, [400010Eh]=A+80h
RST8_37h (ori: RST8_35h)  GetTimer   ;hl=[400010Ch]
RST8_38h (ori: none)      GBA: N/A - Z80:  (?)
Below is undefined/reserved/garbage (values as so in Z80 mode)
(can be used to tweak jap/ori to start GBA-code from inside of Z80-code)
(that, after relocating code to 3000xxxh via DMA via IoWrite function)
RST8_39h (ori: RST8_36h)  bx 0140014h
RST8_3Ah (ori: RST8_37h)  bx 3E700F0h
RST8_3Bh (ori: RST8_38h)  bx 3E70000h+1
RST8_3Ch (ori: RST8_39h)  bx 3E703E6h+1
RST8_3Dh (ori: RST8_3Ah)  bx 3E703E6h+1
RST8_3Eh (ori: RST8_3Bh)  bx 3E703E6h+1
RST8_3Fh (ori: RST8_3Ch)  bx 3E703E6h+1
40h-FFh  (ori: 3Dh-FFh)   bx ...
```

GBA Functions 03xxh (none such in Z80 mode)

```
  RSTX_00h Wait8bit  ;for 16bit: RST0_7Dh
  RSTX_01h GetKeyStateSticky()
  RSTX_02h GetKeyStateRaw()
  RSTX_03h  (?)
  RSTX_04h  (?)
```

GBA Cart e-Reader VPK Decompression

vpk_decompress(src,dest)

```
  collected32bit=80000000h  ;initially empty (endflag in bit31)
  for i=0 to 3, id[i]=read_bits(8), next i, if id[0..3]<>'vpk0' then error
  dest_end=dest+read_bits(32)     ;size of decompressed data (of all strips)
  method=read_bits(8), if method>1 then error
  tree_index=0, read_huffman_tree, disproot=tree_index
  tree_index=tree_index+1, read_huffman_tree, lenroot=tree_index
  ;above stuff is contained only in the first strip. below loop starts at
  ;current location in first strip, and does then continue in further strips.
 decompress_loop:
  if read_bits(1)=0 then                   ;copy one uncompressed data byte,
    [dest]=read_bits(8), dest=dest+1       ;does work without huffman trees
  else
    if disproot=-1 or lenroot=-1 then error  ;compression does require trees
    disp=read_tree(disproot)
    if method=1   ;disp*4 is good for 32bit ARM opcodes
      if disp>2 then disp=disp*4-8 else disp=disp+4*read_tree(disproot)-7
    len=read_tree(lenroot)
    if len=0 or disp<=0 or dest+len-1>dest_end then error ;whoops
    for j=1 to len, [dest]=[dest-disp], dest=dest+1, next j
  if dest<dest_end then decompress_loop
  ret
```

read_bits(num)

```
  mov  data=0
  for i=1 to num
    shl collected32bit,1   ;move next bit to carry, or set zeroflag if empty
    if zeroflag
      collected32bit=[src+0]*1000000h+[src+1]*10000h+[src+2]*100h+[src+3]
      src=src+4            ;read data in 32bit units, in reversed byte-order
      carryflag=1          ;endbit
      rcl collected32bit,1 ;move bit31 to carry (and endbit to bit0)
    rcl data,1             ;move carry to data
  next i
  ret(data)
```

read_tree(root_index)

```
  i=root_index
  while node[i].right<>-1  ;loop until reaching data node
    if read_bits(1)=1 then i=node[i].right else i=node[i].left
  i=node[i].left           ;get number of bits
  i=read_bits(i)           ;read that number of bits
  ret(i)                   ;return that value
```

load_huffman_tree

```
  stacktop=sp
  if read_bits(1)=1 then tree_index=-1, ret  ;exit (empty)
  node[tree_index].right=-1                  ;indicate data node
  node[tree_index].left=read_bits(8)         ;store data value
  if read_bits(1)=1 then ret                 ;exit (only 1 data node at root)
  push tree_index                     ;save previous (child) node
  tree_index=tree_index+1
  jmp data_injump
```

```
load_loop:
 push tree_index                    ;save previous (child) node
 tree_index=tree_index+1
 if read_bits(1)=1 then parent_node
data_injump:
 node[tree_index].right=-1          ;indicate data node
 node[tree_index].left=read_bits(8) ;store data value
 jmp load_loop
parent_node:
 pop node[tree_index].right         ;store 1st child
 pop node[tree_index].left          ;store 2nd child
 if sp<>stacktop then jmp load_loop
 if read_bits(1)=0 then error       ;end bit (must be 1)
 ret
```

The best values for the huffman trees that I've found are 6,9,12-bit displacements for method 0 (best for NES/Z80 code), and two less for method 1, ie. 4,7,10-bit (best for GBA code). And 2,4,10-bit for the length values. The smallest value in node 0, and the other values in node 10 and 11.

Notes

The decompression works similar to the GBA BIOS'es LZ77 decompression function, but without using fixed bit-widths of length=4bit and displacement=12bit, instead, the bit-widths are read from huffman trees (which can also define fixed bit-widths; if data is located directly in the root node).
Unlike the GBA BIOS'es Huffman decompression function, the trees are starting with data entries, end are ending with the root entry. The above load function deciphers the data, and returns the root index.
With the variable bit-widths, the VPK compression rate is quite good, only, it's a pity that the length/disp values are zero-based, eg. for 2bit and 4bit lengths, it'd be much better to assign 2bit as 2..5, and 4bit as 6..21.

Non-VPK

The e-Reader additionally supports an alternate decompression function, indicated by the absence of the "vpk0" ID, which supports compression of increasing byte-values, which isn't useful for program code.
Bit15 of the VPK Size value seems to disable (de-)compression, the VPK Data field is then containing plain uncompressed data.

GBA Cart e-Reader Error Correction

The Error Correction Information that is appended at the end of the Block Header & Data Fragments consists of standard Reed-Solomon codes, which are also used for CD/DVD disks, DSL modems, and digital DVB television signals. That info allows to locate and repair a number of invalid data bytes.

Below code shows how to create and verify error-info (but not how to do the actual error correction). The dtalen,errlen values should be 18h,10h for the Block Header, and 40h,10h for Data Fragments; the latter settings might be possible to get changed to other values though?

append_error_info(data,dtalen,errlen)

```
 reverse_byte_order(data,dtalen)
 zerofill_error_bytes(data,errlen)
 for i=dtalen-1 to errlen  ;loop across data portion
   z = rev[ data[i] xor data[errlen-1] ] ;
   for j=errlen-1 to 0     ;loop across error-info portion
   if j=0 then x=00h else x=data[j-1]
     if z<>FFh then
       y=gg[j], if y<>FFh then
         y=y+z, if y>=FFh then y=y-FFh
         x=x xor pow[y]
     data[j]=x
   next j
 next i
```

```
  invert_error_bytes(data,errlen)
  reverse_byte_order(data,dtalen)
```

verify_error_info(data,dtalen,errlen)
```
  reverse_byte_order(data,dtalen)
  invert_error_bytes(data,errlen)
  make_rev(data,dtalen)
  for i=78h to 78h+errlen-1
    x=0, z=0
    for j=0 to dtalen-1
      y=data[j]
      if y<>FFh then
        y=y+z, if y>=FFh then y=y-FFh
        x=x xor pow[y]
      z=z+i, if z>=FFh then z=z-FFh
    next j
    if x<>0 then error
  next i
  ;(if errors occured, could correct them now)
  make_pow(data,dtalen)
  invert_error_bytes(data,errlen)
  reverse_byte_order(data,dtalen)
```

make_rev(data,len)
```
  for i=0 to len-1, data[i]=rev[data[i]], next i
```

make_pow(data,len)
```
  for i=0 to len-1, data[i]=pow[data[i]], next i
```

invert_error_bytes(data,len)
```
  for i=0 to len-1, data[i]=data[i] xor FFh, next i
```

zerofill_error_bytes(data,len)
```
  for i=0 to len-1, data[i]=00h, next i
```

reverse_byte_order(data,len)
```
  for i=0 to (len-1)/2, x=data[i], data[i]=data[len-i], data[len-i]=x, next i
```

create_pow_and_rev_tables
```
  x=01h, pow[FFh]=00h, rev[00h]=FFh
  for i=00h to FEh
    pow[i]=x, rev[x]=i, x=x*2, if x>=100h then x=x xor 187h
  next i
```

create_gg_table
```
  gg[0]=pow[78h]
  for i=1 to errlen-1
    gg[i]=01h
    for j=i downto 0
      if j=0 then y=00h else y=gg[j-1]
      x=gg[j], if x<>00h then
        x=rev[x]+78h+i, if x>=FFh then x=x-FFh
        y=y xor pow[x]
      gg[j]=y
    next j
  next i
  make_rev(gg,errlen)
```
With above value of 78h, and errlen=10h, gg[00h..0Fh] will be always:
```
  00h,4Bh,EBh,D5h,EFh,4Ch,71h,00h,F4h,00h,71h,4Ch,EFh,D5h,EBh,4Bh
```
So using a hardcoded table should take up less memory than calculating it.

Notes
The actual error correction should be able to fix up to "errlen" errors at known locations (eg. data from blocks that haven't been scanned, or whose 5bit-to-4bit conversion had failed due to an invalid 5bit value), or up to "errlen/2" errors at unknown locations. The corrected data isn't guaranteed to be correct (even if it looks okay to the "verify" function), so the Data Header checksums should be checked, too.

More Info
For more info, I've found Reed-Solomon source code from Simon Rockliff, and an updated version from Robert Morelos-Zaragoza and Hari Thirumoorthy to be useful. For getting started with that source, some important relationships & differences are:

```
pow = alpha_to, but generated as shown above
rev = index_of, dito
b0  = 78h
nn  = dtalen
kk  = dtalen-errlen
%nn = MOD FFh (for the ereader that isn't MOD dtalen)
-1  = FFh
```

And, the ereader processes data/errinfo backwards, starting at the last byte.

GBA Cart e-Reader File Formats

.BMP Files (homebrew 300 DPI strips)
Contains a picture of the whole dotcode strip with address bars and sync marks (see Dotcode chapter) in Microsoft's Bitmap format. The image is conventionally surrounded by a blank 2-pixel border, resulting in a size of 989x44 pixels for long strips. The file should should have 1bit color depth. The pixels per meter entry should match the desired printing resolution, either 300 DPI or 360 DPI. But, resolution of printer hardware is typically specified in inch rather than in meters, so an exact match isn't supported by Microsoft. Most homebrew .BMP files contain nonsense resolutions like 200 DPI, or 300 dots per meter (ca. 8 DPI).

.JPG Files (scanned 1200 DPI strips)
Same as BMP, but should contain a dotcode scanned at 1200 DPI, with correct orientation (the card-edge side at the bottom of the image), and containing only the dotcode (not the whole card), so the JPG size should be about 3450x155 pixels for long strips.
No$gba currently doesn't work with progressive JPGs. Scans with white background can be saved as monochrome JPG. Scans with red/yellow background should contain a correct RED layer (due to the red LED light source) (the brightness of the green/blue layers can be set to zero for better compression).

.RAW Files
Contains the "raw" information from the BMP format, that is, 2-byte block header, 102-byte data, 2-byte block header, 102-byte data, etc. The data portion is interleaved, and includes the full 48-byte data header, titles, vpk compressed data, error-info, and unused bytes. RAW files are excluding Address Bars, Sync Marks, and 4bit-to-5bit encoding.
Each RAW file contains one or more strip(s), so the RAW filesize is either 18*104 bytes (short strip), or 28*104 bytes (long strip), or a multiple thereof (if it contains more than one strip) (although multi-strip games are often stored in separate files for each strip; named file1.raw, file2.raw, etc).

.BIN Files
Filesize should be I*30h, with I=1Ch for short strips, and I=2Ch for long strips, or a multiple thereof (if it contains more than one strip). Each strip consists of the 48-byte Data Header, followed by title(s), and vpk compressed data. Unlike .RAW files, .BIN files aren't interleaved, and do not contain Block Headers, nor error-info, nor unused bytes (in last block). The files do contain padding bytes to match a full strip-size of I*30h.
Caution: Older .BIN files have been using a size-reduced 12-byte header (taken from entries 0Dh, 0Ch, 10h-11h, 26h-2Dh of the 48-byte Data Header; in that order), that files have never contained more than one strip per file, so the filesize should be exactly I*30h-36, the size-reduced header doesn't contain a Primary Type entry, so it's

everyone's bet which Card Type is to be used (hint: the 12-byte headers were based on the assumption that Primary Type would be always 01h on Short Strips, and 02h on Long Strips).

.SAV Files
Contains a copy of the e-Reader's 128Kbyte FLASH memory. With the saved e-Reader application being located in the 2nd 64K-bank, the data consists of a header with title and gba/nes/z80 format info, followed by the vpk compressed data. The FLASH memory does also contain e-Reader calibration settings, the remaining 100Kbytes are typically FFh-filled.

GBA Cart Unknown Devices

GBA Infra-Red Port (AGB-006)
No info?

GBA Cart Protections

Classic NES Series
These are some NES/Famicom games ported or emulated to work on GBA. The games are doing some uncommon stuff that can cause compatibility problems when not using original GBA consoles or cartridges.
- CPU pipeline (selfmodifying code that shall NOT affect prefetched opcodes)
- STMDA write to I/O ports (writes in INCREASING order, not DECREASING order)
- SRAM detection (refuses to run if SRAM exists; the games do contain EEPROM)
- ROM mirrors (instead of the usual increasing numbers in unused ROM area)
- RAM mirrors (eg. main RAM accessed at 2F00000h instead of 2000000h)
Note: These games can be detected by checking [80000ACh]="F" (ie. game code="Fxxx").

GBA Flashcards

Flashcards are re-writable cartridges using FLASH memory, allowing to test even multiboot-incompatible GBA software on real hardware, providing a good development environment when used in combination with a reasonable software debugger.

The carts can be written to from external tools, or directly from GBA programs.
Below are pseudo code flowcharts for detect, erase, and write operations.
All flash reads/writes are meant to be 16bit (ldrh/strh) memory accesses.

detect_flashcard:
```
configure_flashcard(9E2468Ah,9413h)    ;unlock flash advance cards
turbo=1, send_command(8000000h,90h)    ;enter ID mode (both chips, if any)
maker=[8000000h], device=[8000000h+2]
IF maker=device THEN device=[8000000h+4] ELSE turbo=0
flashcard_read_mode                    ;exit ID mode
search (maker+device*10000h) in device_list
total/erase/write_block_size = list_entry SHL turbo
```

flashcard_erase(dest,len):
```
FOR x=1 to len/erase_block_size
 send_command(dest,20h)        ;erase sector command
 send_command(dest,D0h)        ;confirm erase sector
 dest=dest+erase_block_size
IF wait_busy=okay THEN NEXT x
enter_read_mode                ;exit erase/status mode
```

flashcard_write(src,dest,len):

```
siz=write_block_size
FOR x=1 to len/siz
 IF siz=2 THEN send_command(dest,10h)  ;write halfword command
 IF siz>2 THEN send_command(dest,E8h)  ;write to buffer command
 IF siz>2 THEN send_command(dest,16-1) ;buffer size 16 halfwords (per chip)
 FOR y=1 TO siz/2
  [dest]=[src], dest=dest+2, src=src+2 ;write data to buffer
 NEXT y
 IF siz>2 THEN send_command(dest,D0h)  ;confirm write to buffer
IF wait_busy=okay THEN NEXT x
enter_read_mode                        ;exit write/status mode
```

send_command(adr,val):

```
[adr]=val
IF turbo THEN [adr+2]=val
```

enter_read_mode:

```
send_command(8000000h,FFh)     ;exit status mode
send_command(8000000h,FFh)     ;again maybe more stable (as in jeff's source)
```

flashcard_wait_busy:

```
start=time
REPEAT
 stat=[8000000h] XOR 80h
 IF turbo THEN stat=stat OR ([8000000h+2] XOR 80h)
 IF (stat AND 7Fh)>0 THEN error
 IF (stat AND 80h)=0 THEN ready
 IF time-start>5secs THEN timeout
UNTIL ready OR error OR timeout
IF error OR timeout THEN send_command(8000000h,50h)    ;clear status
```

configure_flashcard(adr,val): ;required for Flash Advance cards only

```
[930ECA8h]=5354h
[802468Ah]=1234h, repeated 500 times
[800ECA8h]=5354h
[802468Ah]=5354h
[802468Ah]=5678h, repeated 500 times
[930ECA8h]=5354h
[802468Ah]=5354h
[8ECA800h]=5678h
[80268A0h]=1234h
[802468Ah]=ABCDh, repeated 500 times
[930ECA8h]=5354h
[adr]=val
```

init_backup: ;no info how to use that exactly

```
configure_flashcard(942468Ah,???)
```

device_list: (id code, total/erase/write sizes in bytes)

ID Code	Total	Erase	Write	Name
-??-00DCh	?	?	?	Hudson Cart (???)
00160089h	4M	128K	32	Intel i28F320J3A (Flash Advance)
00170089h	8M	128K	32	Intel i28F640J3A (Flash Advance)
00180089h	16M	128K	32	Intel i28F128J3A (Flash Advance)
00E200B0h	?	64K	2	Sharp LH28F320BJE ? (Nintendo)

Notes

All flashcards should work at 4,2 waitstates (power on default), most commercial games change waits to 3,1 which may work unstable with some/older FA flashcards. Intel FLASH specified to have a lifetime of 100,000

erases, and average block erase time 1 second (up to 5 second in worst cases).
Aside from the main FLASH memory, Flash Advance (FA) (aka Visoly) cards additionally contain battery buffered SRAM backup, and FLASH backup, and in some cases also EEPROM backup.
Turbo FA cards are containing two chips interlaced (at odd/even halfword addresses), allowing to write/erase both chips simultaneously, resulting in twice as fast programming time.
Standard Nintendo flash carts have to be modified before you can actually write to them. This is done by removing resistor R7 and putting it at empty location R8.
Mind that write/erase/detect modes output status information in ROM area, so that in that modes all GBA program code (and any interrupt handlers) must be executed in WRAM, not in ROM.

Thanks to Jeff Frohwein for his FAQ and CARTLIB sample in FLGBA at devrs.com

GBA Cheat Devices

Codebreaker (US) aka Xploder (EUR).
Gameshark (US) aka Action Replay (EUR).

GBA Cheat Codes - General Info

Cheat devices are external adapters, connected between the GBA and the game cartridge. The devices include a BIOS ROM which is, among others, used to prompt the user to enter cheat codes.
These codes are used to patch specified memory locations for a certain GBA game, allowing the user to gain goodies such like Infinite sex, 255 Cigarettes, etc.

ROM and RAM Patches
For ROM Patches, the device watches the address bus, if it matches a specified address then it outputs a patched value to the data bus, that mechanism is implemented by hardware, aside from the Hook Enable Code some devices also allow a limited number of cheats to use ROM patches.
Most cheat codes are RAM patches, each time when the hook procedure is executed it will process all codes and overwrite the specified addresses in RAM (or VRAM or I/O area) by the desired values.

Enable Codes (Must Be On)
Enable codes usually consist of the Game ID, Hook Address, and eventually a third code used to encrypt all following codes. The Game ID is used to confirm that the correct cartridge is inserted, just a verification, though the device may insist on the ID code.
The Hook Address specifies an address in cartridge ROM, and should point to an opcode which is executed several times per second (eg. once per frame, many codes place the hook in the joypad handler). At the hook address, the device redirects to its own BIOS, processes the RAM patches, and does then return control to the game cartridge.
Note: The hook address should not point to opcodes with relative addressing (eg. B, BL, LDR Rd,=Imm, ADD Rd,=Imm opcodes - which are all relative to PC program counter register).

Alignment
Addresses for 16bit or 32bit values should be properly aligned.

GBA Cheat Codes - Codebreaker/Xploder

Codebreaker Codes

```
0000xxxx 000y  Enable Code 1 - Game ID
1aaaaaaa 000z  Enable Code 2 - Hook Address
2aaaaaaa yyyy  [aaaaaaa]=[aaaaaaa] OR yyyy
3aaaaaaa 00yy  [aaaaaaa]=yy
4aaaaaaa yyyy  [aaaaaaa+0..(cccc-1)*ssss]=yyyy+0..(cccc-1)*ssss
iiiicccc ssss  parameters for above code
5aaaaaaa cccc  [aaaaaaa+0..(cccc-1)]=11,22,33,44,etc.
11223344 5566  parameter bytes 1..6 for above code (example)
77880000 0000  parameter bytes 7..8 for above code (padded with zero)
6aaaaaaa yyyy  [aaaaaaa]=[aaaaaaa] AND yyyy
7aaaaaaa yyyy  IF [aaaaaaa]=yyyy THEN (next code)
8aaaaaaa yyyy  [aaaaaaa]=yyyy
9xyyxxxx xxxx  Enable Code 0 - Encrypt all following codes (optional)
Aaaaaaaa yyyy  IF [aaaaaaa]<>yyyy THEN (next code)
Baaaaaaa yyyy  IF [aaaaaaa]>yyyy THEN (next code) (signed comparison)
Caaaaaaa yyyy  IF [aaaaaaa]<yyyy THEN (next code) (signed comparison)
D0000020 yyyy  IF [joypad] AND yyyy = 0 THEN (next code)
Eaaaaaaa yyyy  [aaaaaaa]=[aaaaaaa]+yyyy
Faaaaaaa yyyy  IF [aaaaaaa] AND yyyy THEN (next code)
```

Codebreaker Enable Codes

Hook Address 'aaaaaaa' is a 25bit offset in ROM-image (0-1FFFFFFh).
Flag byte 'y' (usually 0Ah), Bit1=Disable IRQs, Bit3=CRC Exists.
Code Handler Store Address 'z' (0-7, usually 7) (8000100h+z*400000h).
Checksum 'xxxx' for first 64Kbytes of cartridge (no$gba pads by FFh if ROM is smaller than 64K). Calculated, by using unsigned 16bit values, as such:

```
crc=FFFFh
for i=0 to FFFFh
 x=byte[i] xor (crc/100h)
 x=x xor (x/10h)
 crc=(crc*100h) xor (x*1001h) xor (x*20h)
next i
```

Codebreaker Encryption

codebreaker_change_encryption:
Encryption can be (optionally) activated by code "9xyyxxxx xxxx",

```
for i=0 to 2Fh, swaplist[i]=i, next i
randomizer = 1111h xor byte[code+4]                              ;LSB value
for i=0 to 4Fh
  exchange swaplist[random MOD 30h] with swaplist[random MOD 30h]
next i
halfword[seedlist+0] = halfword[code+0]                          ;LSW address
randomizer = 4EFAD1C3h
for i=0 to byte[code+3]-91h, randomizer=random, next i           ;MSB address
word[seedlist+2]=random, halfword[seedlist+6]=random
randomizer = F254h xor byte[code+5]                              ;MSB value
for i=0 to byte[code+5]-01h, randomizer=random, next i           ;MSB value
word[seedlist+8]=random, halfword[seedlist+12]=random
;note: byte[code+2] = don't care
ret
```

The above random function works like so:

```
randomizer=randomizer*41C64E6Dh+3039h, x=(randomizer SHL 14 AND C0000000h)
randomizer=randomizer*41C64E6Dh+3039h, x=(randomizer SHR 1  AND 3FFF8000h)+x
randomizer=randomizer*41C64E6Dh+3039h, x=(randomizer SHR 16 AND 00007FFFh)+x
return(x)
```

Once when encryption is activated, all following codes are decrypted like so:

```
for i=2Fh to 0
  j=swaplist[i]
```

```
    bitno1=(i AND 7), index1=xlatlist[i/8]
    bitno2=(j AND 7), index2=xlatlist[j/8]
    exchange [code+index1].bitno1 with [code+index2].bitno2
  next i
  word[code+0] = word[code+0] xor word[seedlist+8]
  i = (byte[code+3]*1010000h + byte[code+0]*100h + byte[code+5])
  i = (halfword[code+1]*10001h) xor (word[seedlist+2]) xor i
  i = (byte[seedlist+0]*1010101h) xor (byte[seedlist+1]*1000000h) xor i
  j = (byte[code+5] + (byte[code+0] xor byte[code+4])*100h)
  j = (byte[seedlist+0]*101h) xor halfword[seedlist+6] xor j
  word[code+0] = i, halfword[code+4] = j
```
The above xlatlist is fixed: xlatlist[0..5] = 3,2,1,0,5,4

GBA Cheat Codes - Gameshark/Action Replay V1/V2

Gameshark RAW Codes (These codes must be encrypted before using them)

```
  0aaaaaaa 000000xx  [aaaaaaa]=xx
  1aaaaaaa 0000xxxx  [aaaaaaa]=xxxx
  2aaaaaaa xxxxxxxx  [aaaaaaa]=xxxxxxxx
  3000cccc xxxxxxxx  write xxxxxxxx to (cccc-1) addresses (list in next codes)
  aaaaaaaa aaaaaaaa  parameter for above code, containing two addresses each
  aaaaaaaa 00000000  last parameter for above, zero-padded if only one address
  60aaaaaa y000xxxx  [8000000h+aaaaaa*2]=xxxx (ROM Patch)
  8a1aaaaa 000000xx  IF GS_Button_Down THEN [a0aaaaa]=xx
  8a2aaaaa 0000xxxx  IF GS_Button_Down THEN [a0aaaaa]=xxxx
  80F00000 0000xxxx  IF GS_Button_Down THEN slowdown xxxx * ? cycles per hook
  Daaaaaaa 0000xxxx  IF [aaaaaaa]=xxxx THEN (next code)
  E0zzxxxx 0aaaaaaa  IF [aaaaaaa]=xxxx THEN (next 'zz' codes)
  Faaaaaaa 00000x0y  Enable Code - Hook Routine
  xxxxxxxx 001DC0DE  Enable Code - Game Code ID (value at [0ACh] in cartridge)
  DEADFACE 0000xxyy  Change Encryption Seeds
```

Enable Code - Hook Routine

Hook Address 'aaaaaaa' is a 28bit ROM address (8FFFFFFh-9FFFFFFh).
Used to insert the GS code handler routine where it will be executed at least 20 times per second. Without this code, GSA can not write to RAM.

```
 y=1 - Executes code handler without backing up the LR register.
 y=2 - Executes code handler and backs up the LR register.
 y=3 - Replaces a 32-bit pointer used for long-branches.
 x=0 - Must turn GSA off before loading game.
 x=1 - Must not do that.
```

ROM Patch

This type allows GSA to intercept ROM reads and returns the value xxxx.

```
 y=0 wait for the code handler to enable the patch
 y=1 patch is enabled before the game starts
 y=2 unknown ?
```

Note: V1/V2 hardware can only have up to 1 user-defined rom patch max. V3 can have up to 4. Some enable code types can shorten the amount of user-defined rom patches available.

Gameshark Encryption

A=Left half, and V=Right half of code.

```
  FOR I=1 TO 32
    A=A + (V*16+S0) XOR (V+I*9E3779B9h) XOR (V/32+S1)
    V=V + (A*16+S2) XOR (A+I*9E3779B9h) XOR (A/32+S3)
  NEXT I
```

Upon startup, the initial encryption seeds are:

```
  S0=09F4FBBDh S1=9681884Ah S2=352027E9h S3=F3DEE5A7h
```

Upon DEADFACE 0000xxyy, the S0..S3 seeds are changed like so:

```
FOR y=0 TO 3
 FOR x=0 TO 3
  z = T1[(xx+x) AND FFh] + T2[(yy+y) AND FFh]
  Sy = Sy*100h + (z AND FFh)
 NEXT x
NEXT y
```

All calculations truncated to unsigned 32bit integer values.
T1 and T2 are translation tables contained in the gameshark cartridge.

GBA Cheat Codes - Pro Action Replay V3

Pro Action Replay V3 - RAW Codes

```
C4aaaaaa 0000yyyy  Enable Code - Hook Routine at [8aaaaaa]
xxxxxxxx 001DC0DE  Enable Code - ID Code [080000AC]
DEADFACE 0000xxxx  Enable Code - Change Encryption Seeds
00aaaaaa xxxxxxyy  [a0aaaaa..a0aaaaa+xxxxxx]=yy
02aaaaaa xxxxyyyy  [a0aaaaa..a0aaaaa+xxxx*2]=yyyy
04aaaaaa yyyyyyyy  [a0aaaaa]=yyyyyyyy
40aaaaaa xxxxxxyy  [ [a0aaaaa] + xxxxxx ]=yy   (Indirect)
42aaaaaa xxxxyyyy  [ [a0aaaaa] + xxxx*2 ]=yyyy (Indirect)
44aaaaaa yyyyyyyy  [ [a0aaaaa] ]=yyyyyyyy      (Indirect)
80aaaaaa 000000yy  [a0aaaaa]=[a0aaaaa]+yy
82aaaaaa 0000yyyy  [a0aaaaa]=[a0aaaaa]+yyyy
84aaaaaa yyyyyyyy  [a0aaaaa]=[a0aaaaa]+yyyyyyyy
C6aaaaaa 0000yyyy  [4aaaaaa]=yyyy              (I/O Area)
C7aaaaaa yyyyyyyy  [4aaaaaa]=yyyyyyyy          (I/O Area)
iiaaaaaa yyyyyyyy  IF [a0aaaaa] <cond> <value> THEN <action>
00000000 60000000  ELSE (?)
00000000 40000000  ENDIF (?)
00000000 0800xx00  AR Slowdown : loops the AR xx times
00000000 00000000  End of the code list
00000000 10aaaaaa 000000zz 00000000  IF AR_BUTTON THEN [a0aaaaa]=zz
00000000 12aaaaaa 0000zzzz 00000000  IF AR_BUTTON THEN [a0aaaaa]=zzzz
00000000 14aaaaaa zzzzzzzz 00000000  IF AR_BUTTON THEN [a0aaaaa]=zzzzzzzz
00000000 18aaaaaa 0000zzzz 00000000  [8000000+aaaaaa*2]=zzzz  (ROM Patch 1)
00000000 1Aaaaaaa 0000zzzz 00000000  [8000000+aaaaaa*2]=zzzz  (ROM Patch 2)
00000000 1Caaaaaa 0000zzzz 00000000  [8000000+aaaaaa*2]=zzzz  (ROM Patch 3)
00000000 1Eaaaaaa 0000zzzz 00000000  [8000000+aaaaaa*2]=zzzz  (ROM Patch 4)

00000000 80aaaaaa 000000yy ssccssss  repeat cc times [a0aaaaa]=yy
 (with yy=yy+ss, a0aaaaa=a0aaaaa+ssss after each step)

00000000 82aaaaaa 0000yyyy ssccssss  repeat cc times [a0aaaaa]=yyyy
 (with yyyy=yyyy+ss, a0aaaaa=a0aaaaa+ssss*2 after each step)

00000000 84aaaaaa yyyyyyyy ssccssss  repeat cc times [a0aaaaa]=yyyyyyyy
 (with yyyy=yyyy+ss, a0aaaaa=a0aaaaa+ssss*4 after each step)
```

Warning: There is a bug on the real AR (v2 upgraded to v3, and maybe on real v3) with the 32bit Increment Slide code. You HAVE to add a code (best choice is 80000000 00000000 : add 0 to value at address 0) right after it, else the AR will erase the 2 last 8 digits lines of the 32 Bits Inc. Slide code when you enter it !!!

Final Notes
The 'turn off all codes' makes an infinite loop (that can't be broken, unless the condition becomes True). - How? By Interrupt? Huh?
ROM Patch1 works on real V3 and, on V1/V2 upgraded to V3.
ROM Patch2,3,4 work on real V3 hardware only.

Pro Action Replay V3 Conditional Codes - iiaaaaaa yyyyyyyy
The 'ii' is composed of <cond> + <value> + <action>.

```
<cond>           <value>           <action>
08 Equal =       00 8bit zz        00 execute next code
10 Not equal <>  02 16bit zzzz     40 execute next two codes
18 Signed <      04 32bit zzzzzzzz 80 execute all following
20 Signed >      06 (always false)    codes until ELSE or ENDIF
28 Unsigned <                      C0 normal ELSE turn off all codes
30 Unsigned >
38 Logical AND
```

For example, ii=18h+02h+40h=5Ah, produces IF [a0aaaaa]<zzzz THEN next 2 codes.

Always... Codes

```
For the "Always..." codes:
- XXXXXXXX can be any authorised address except 00000000 (eg. use 02000000).
- ZZZZZZZZ can be anything.
- The "y" in the code data must be in the [1-7] range (which means not 0).
typ=y,sub=0,siz=3   Always skip next line.
typ=y,sub=1,siz=3   Always skip next 2 lines.
typ=y,sub=2,siz=3   Always Stops executing all the codes below.
typ=y,sub=3,siz=3   Always turn off all codes.
```

Code Format (ttaaaaaa xxxxyyzz)

```
adr mask = 003FFFFF
n/a mask = 00C00000 ;not used
xtr mask = 01000000 ;used only by I/O write, and MSB of Hook
siz mask = 06000000
typ mask = 38000000 ;0=normal, other=conditional
sub mask = C0000000
```

Pro Action Replay V3 Encryption
Works exactly as for Gameshark Encryption, but with different initial seeds,

```
S0=7AA9648Fh S1=7FAE6994h S2=C0EFAAD5h S3=42712C57h
```

And, the T1 and T2 translation tables are different, too.

GBA Gameboy Player

The Gameboy Player is an "adapter" for the Gamecube console. It's basicly is a GBA in a black box without LCD screen and without buttons, connected to an expansion port at the bottom of the Gamecube. The Gamecube is then capturing the GBA video output (and passing it to the television set), and in the other direction, passing the Gamecube joypad input to the GBA inputs.

Unlocking and Detecting Gameboy Player Functions
Both unlocking and detection requires to display the 240x160 pixel Gameboy Player logo (44 colors) for a number of frames... maybe at least 3-4 frames? not sure if it checks the color of the logo... so maybe it can be hidden by using dark gray on black background?
While displaying this logo, the joypad data will switch between values 03FFh (2 frames duration) and 030Fh (1 frame duration). The latter value (left, right, up, down all pressed) indicates that it's a Gameboy Player.

Palette
Knowing Nintendo, they've probably not reproduced the blurred GBA colors (?), so the games won't look as desired on the TV screen. Unless the game does detect the Gameboy Player, and adjust the colors accordingly by software.

Rumble
The only known existing special function is the joypad rumble function, controlled by sending data through the

serial port (the normal GBA port, even though it also has the connectors).

The Game Boy Player added a rumble feature to certain Game Boy Advance games when played with a GameCube controller. Those games included:

```
Drill Dozer (supports BOTH handheld-rumble and GBP-rumble?)
Mario & Luigi: Superstar Saga
Pokemon Pinball: Ruby & Sapphire
Shikakui Atama wo Marukusuru Advance: Kokugo Sansu Rika Shakai
Shikakui Atama wo Marukusuru Advance: Kanji Keisan
Summon Night Craft Sword Monogatari: Hajimari no Ishi
Super Mario Advance 4: Super Mario Bros. 3
```

Fredrik Olsson (aka Flubba) has implemented rumble in 3 applications now RumblePong (FluBBA) (homebrew)

```
Remudvance (FluBBA) (homebrew)
Goomba (FluBBA) (8bit Gameboy Color Emulator for 32bit GBA) (homebrew)
and, supposedly in "Tetanus on Drugs" (Tepples) (homebrew)
```

The GBP can also use some of the extra controllers for the GC like the Bongas from Donkey Konga.

The logo requires at least 256 colors, it doesn't matter if you use a tiled screen mode or a bitmapped one, the logo can be ripped from either "Pokemon Pinball" or "Super Mario Advance 4".

Rumble

After detecting/unlocking the Gameboy Player, init RCNT and SIOCNT to 32bit normal mode, external clock, SO=high, with IRQ enabled, and set the transfer start bit. You should then receive the following sequence (about once per frame), and your serial IRQ handler should send responses accordingly:

```
Receive  Response
0000494E 494EB6B1
xxxx494E 494EB6B1
B6B1494E 544EB6B1
B6B1544E 544EABB1
ABB1544E 4E45ABB1
ABB14E45 4E45B1BA
B1BA4E45 4F44B1BA
B1BA4F44 4F44B0BB
B0BB4F44 8000B0BB
B0BB8002 10000010
10000010 20000013
20000013 40000004
30000003 40000004
30000003 40000004
30000003 40000004
30000003 400000yy
30000003 40000004
```

The first part of the transfer just contains the string "NINTENDO" split into 16bit fragments, and bitwise inversions thereof (eg. 494Eh="NI", and B6B1h=NOT 494Eh). In the second part, <yy> should be 04h=RumbleOff, or 26h=RumbleOn.

Note

If it's having a similar range of functions as the 8bit Super Gameboy, then the Gameboy Player might be also able to access analogue joypad input, and to access other features of the Gamecube hardware, up to possibly executing code on the Gamecube CPU...?

GBA Unpredictable Things

Forward
Most of the below is caused by 'traces' from previous operations which have used the databus. No promises that the results are stable on all current or future GBA models, and/or under all temperature and interference circumstances.
Also, below specifies 32bit data accesses only. When reading units less than 32bit, data is rotated depending on the alignment of the originally specified address, and 8bit or 16bit are then isolated from the 32bit value as usually.

Reading from BIOS Memory (00000000-00003FFF)
The BIOS memory is protected against reading, the GBA allows to read opcodes or data only if the program counter is located inside of the BIOS area. If the program counter is not in the BIOS area, reading will return the most recent successfully fetched BIOS opcode (eg. the opcode at [00DCh+8] after startup and SoftReset, the opcode at [0134h+8] during IRQ execution, and opcode at [013Ch+8] after IRQ execution, and opcode at [0188h+8] after SWI execution).

Reading from Unused Memory (00004000-01FFFFFF,10000000-FFFFFFFF)
Accessing unused memory at 00004000h-01FFFFFFh, and 10000000h-FFFFFFFFh (and 02000000h-03FFFFFFh when RAM is disabled via Port 4000800h) returns the recently pre-fetched opcode. For ARM code this is simply:
```
  WORD = [$+8]
```
For THUMB code the result consists of two 16bit fragments and depends on the address area and alignment where the opcode was stored.
For THUMB code in Main RAM, Palette Memory, VRAM, and Cartridge ROM this is:
```
  LSW = [$+4], MSW = [$+4]
```
For THUMB code in BIOS or OAM (and in 32K-WRAM on Original-NDS (in GBA mode)):
```
  LSW = [$+4], MSW = [$+6]   ;for opcodes at 4-byte aligned locations
  LSW = [$+2], MSW = [$+4]   ;for opcodes at non-4-byte aligned locations
```
For THUMB code in 32K-WRAM on GBA, GBA SP, GBA Micro, NDS-Lite (but not NDS):
```
  LSW = [$+4], MSW = OldHI   ;for opcodes at 4-byte aligned locations
  LSW = OldLO, MSW = [$+4]   ;for opcodes at non-4-byte aligned locations
```
Whereas OldLO/OldHI are usually:
```
  OldLO=[$+2], OldHI=[$+2]
```
Unless the previous opcode's prefetch was overwritten; that can happen if the previous opcode was itself an LDR opcode, ie. if it was itself reading data:
```
  OldLO=LSW(data), OldHI=MSW(data)
  Theoretically, this might also change if a DMA transfer occurs.
```
Note: Additionally, as usually, the 32bit data value will be rotated if the data address wasn't 4-byte aligned, and the upper bits of the 32bit value will be masked in case of LDRB/LDRH reads.
Note: The opcode prefetch is caused by the prefetch pipeline in the CPU itself, not by the external gamepak prefetch, ie. it works for code in ROM and RAM as well.

Reading from Unused or Write-Only I/O Ports
Works like above Unused Memory when the entire 32bit memory fragment is Unused (eg. 0E0h) and/or Write-Only (eg. DMA0SAD). And otherwise, returns zero if the lower 16bit fragment is readable (eg. 04Ch=MOSAIC, 04Eh=NOTUSED/ZERO).

Reading from GamePak ROM when no Cartridge is inserted
Because Gamepak uses the same signal-lines for both 16bit data and for lower 16bit halfword address, the entire gamepak ROM area is effectively filled by incrementing 16bit values (Address/2 AND FFFFh).

Memory Mirrors
Most internal memory is mirrored across the whole 24bit/16MB address space in which it is located: Slow On-board RAM at 2XXXXXX, Fast On-Chip RAM at 3XXXXXXh, Palette RAM at 5XXXXXXh, VRAM at 6XXXXXXh, and OAM at 7XXXXXXh. Even though VRAM is sized 96K (64K+32K), it is repeated in steps

of 128K (64K+32K+32K, the two 32K blocks itself being mirrors of each other).
BIOS ROM, Normal ROM Cartridges, and I/O area are NOT mirrored, the only exception is the undocumented I/O port at 4000800h (repeated each 64K).
The 64K SRAM area is mirrored across the whole 32MB area at E000000h-FFFFFFFh, also, inside of the 64K SRAM field, 32K SRAM chips are repeated twice.

Writing 8bit Data to Video Memory
Video Memory (BG, OBJ, OAM, Palette) can be written to in 16bit and 32bit units only. Attempts to write 8bit data (by STRB opcode) won't work:
Writes to OBJ (6010000h-6017FFFh) (or 6014000h-6017FFFh in Bitmap mode) and to OAM (7000000h-70003FFh) are ignored, the memory content remains unchanged.
Writes to BG (6000000h-600FFFFh) (or 6000000h-6013FFFh in Bitmap mode) and to Palette (5000000h-50003FFh) are writing the new 8bit value to BOTH upper and lower 8bits of the addressed halfword, ie. "[addr AND NOT 1]=data*101h".

Using Invalid Tile Numbers
In Text mode, large tile numbers (combined with a non-zero character base setting in BGnCNT register) may exceed the available 64K of BG VRAM.
On GBA and GBA SP, such invalid tiles are displayed as if the character data is filled by the 16bit BG Map entry value (ie. as vertically striped tiles). Above applies only if there is only one BG layer enabled, with two or more layers, things are getting much more complicated: tile-data is then somehow derived from the other layers, depending on their priority order and scrolling offsets.
On NDS (in GBA mode), such invalid tiles are displayed as if the character data is zero-filled (ie. as invisible/transparent tiles).

Accessing SRAM Area by 16bit/32bit
Reading retrieves 8bit value from specified address, multiplied by 0101h (LDRH) or by 01010101h (LDR).
Writing changes the 8bit value at the specified address only, being set to LSB of (source_data ROR (address*8)).

NDS Reference

Overview

Hardware Programming

Other

DS Technical Data

Processors

```
1x ARM946E-S 32bit RISC CPU, 66MHz (NDS9 video) (not used in GBA mode)
1x ARM7TDMI  32bit RISC CPU, 33MHz (NDS7 sound) (16MHz in GBA mode)
```

Internal Memory

```
4096KB Main RAM (8192KB in debug version)
96KB   WRAM (64K mapped to NDS7, plus 32K mappable to NDS7 or NDS9)
60KB   TCM/Cache (TCM: 16K Data, 32K Code) (Cache: 4K Data, 8K Code)
656KB  VRAM (allocateable as BG/OBJ/2D/3D/Palette/Texture/WRAM memory)
4KB    OAM/PAL (2K OBJ Attribute Memory, 2K Standard Palette RAM)
248KB  Internal 3D Memory (104K Polygon RAM, 144K Vertex RAM)
?KB    Matrix Stack, 48 scanline cache
8KB    Wifi RAM
256KB  Firmware FLASH (512KB in iQue variant, with chinese charset)
36KB   BIOS ROM (4K NDS9, 16K NDS7, 16K GBA)
```

Video

```
2x LCD screens (each 256x192 pixel, 3 inch, 18bit color depth, backlight)
2x 2D video engines (extended variants of the GBA's video controller)
1x 3D video engine (can be assigned to upper or lower screen)
1x video capture (for effects, or for forwarding 3D to the 2nd 2D engine)
```

Sound

```
16 sound channels (16x PCM8/PCM16/IMA-ADPCM, 6x PSG-Wave, 2x PSG-Noise)
2 sound capture units (for echo effects, etc.)
Output: Two built-in stereo speakers, and headphones socket
Input:  One built-in microphone, and microphone socket
```

Controls

```
Gamepad      4 Direction Keys, 8 Buttons
Touchscreen  (on lower LCD screen)
```

Communication Ports

```
Wifi IEEE802.11b
```

Specials

```
Built-in Real Time Clock
Power Managment Device
Hardware divide and square root functions
CP15 System Control Coprocessor (cache, tcm, pu, bist, etc.)
```

External Memory

```
NDS Slot (for NDS games) (encrypted 8bit data bus, and serial 1bit bus)
GBA Slot (for NDS expansions, or for GBA games) (but not for DMG/CGB games)
```

Manufactured Cartridges

```
ROM: 16MB, 32MB, or 64MB
EEPROM/FLASH/FRAM: 0.5KB, 8KB, 64KB, 256KB, or 512KB
```

Can be booted from

```
NDS Cartridge (NDS mode)
Firmware FLASH (NDS mode) (eg. by patching firmware via ds-xboo cable)
Wifi (NDS mode)
GBA Cartridge (GBA mode) (without DMG/CGB support) (without SIO support)
```

Power Supply

```
Built-in rechargeable Lithium ion battery, 3.7V 1000mAh (DS-Lite)
External Supply: 5.2V DC
```

NDS-Lite

Slightly smaller than the original NDS, coming in a more decently elegant case. The LCDs are much more colorful (and thus not backwards compatible with any older NDS or GBA games), and the LCDs support wider viewing angles. Slightly different power managment device (with selectable backlight brightness, new external power source flag, lost audio amplifier mute flag). Slightly different Wifi controller (different chip ID, different dirt effects when accessing invalid wifi ports and unused wifi memory regions, different behaviour on GAPDISP registers, RF/BB chips replaced by a single chip). Slightly different touch screen controller (with new unused input, and slightly different powerdown bits).

Notice
NDS9 means the ARM9 processor and its memory and I/O ports in NDS mode
NDS7 means the ARM7 processor and its memory and I/O ports in NDS mode
GBA means the ARM7 processor and its memory and I/O ports in GBA mode

The two Processors
Most game code is usually executed on the ARM9 processor (in fact, Nintendo reportedly doesn't allow developers use the ARM7 processor, except by predefined API functions, anyways, even with the most likely inefficient API code, most of the ARM7's 33MHz horsepower is left unused).
The ARM9's 66MHz "horsepower" is a different tale - it seems Nintendo thought that a 33MHz processor would be too "slow" for 3D games, and so they (tried to) badge an additional CPU to the original GBA hardware. However, the real 66MHz can be used only with cache and tcm, all other memory and I/O accesses are delayed to the 33MHz bus clock, that'd be still quite fast, but, there seems to be a hardware glitch that adds 3 waitcycles to all nonsequential accesses at the NDS9 side, which effectively drops its bus clock to about 8MHz, making it ways slower than the 33MHz NDS7 processor, it's even slower than the original 16MHz GBA processor. Altogether, with the bugged 66MHz, and the unused 33MHz, Nintendo could have reached almost the same power when staying with the GBA's 16MHz processor :-)
Although, when properly using cache/tcm, then the 66MHz processor <can> be very fast, still, the NDS should have worked as well with a single processor, though using only an ARM9 might cause a lot of compatibility problems with GBA games, so there's at least one reason for keeping the ARM7 included.

DS I/O Maps

ARM9 I/O Map
ARM9 Display Engine A
```
  4000000h  4    2D Engine A - DISPCNT - LCD Control (Read/Write)
  4000004h  2    2D Engine A+B - DISPSTAT - General LCD Status (Read/Write)
  4000006h  2    2D Engine A+B - VCOUNT - Vertical Counter (Read only)
  4000008h  50h  2D Engine A (same registers as GBA, some changed bits)
  4000060h  2    DISP3DCNT - 3D Display Control Register (R/W)
  4000064h  4    DISPCAPCNT - Display Capture Control Register (R/W)
  4000068h  4    DISP_MMEM_FIFO - Main Memory Display FIFO (R?/W)
  400006Ch  2    2D Engine A - MASTER_BRIGHT - Master Brightness Up/Down
```
GBA I/O Map
ARM9 DMA, Timers, and Keypad
```
  40000B0h  30h  DMA Channel 0..3
  40000E0h  10h  DMA FILL Registers for Channel 0..3
  4000100h  10h  Timers 0..3
  4000130h  2    KEYINPUT
  4000132h  2    KEYCNT
```
ARM9 IPC/ROM
```
  4000180h  2  IPCSYNC - IPC Synchronize Register (R/W)
  4000184h  2  IPCFIFOCNT - IPC Fifo Control Register (R/W)
  4000188h  4  IPCFIFOSEND - IPC Send Fifo (W)
  40001A0h  2  AUXSPICNT - Gamecard ROM and SPI Control
  40001A2h  2  AUXSPIDATA - Gamecard SPI Bus Data/Strobe
  40001A4h  4  Gamecard bus timing/control
  40001A8h  8  Gamecard bus 8-byte command out
  40001B0h  4  Gamecard Encryption Seed 0 Lower 32bit
  40001B4h  4  Gamecard Encryption Seed 1 Lower 32bit
  40001B8h  2  Gamecard Encryption Seed 0 Upper 7bit (bit7-15 unused)
  40001BAh  2  Gamecard Encryption Seed 1 Upper 7bit (bit7-15 unused)
```
ARM9 Memory and IRQ Control
```
  4000204h  2  EXMEMCNT - External Memory Control (R/W)
  4000208h  2  IME - Interrupt Master Enable (R/W)
  4000210h  4  IE  - Interrupt Enable (R/W)
  4000214h  4  IF  - Interrupt Request Flags (R/W)
```

```
  4000240h  1  VRAMCNT_A - VRAM-A (128K) Bank Control (W)
  4000241h  1  VRAMCNT_B - VRAM-B (128K) Bank Control (W)
  4000242h  1  VRAMCNT_C - VRAM-C (128K) Bank Control (W)
  4000243h  1  VRAMCNT_D - VRAM-D (128K) Bank Control (W)
  4000244h  1  VRAMCNT_E - VRAM-E (64K) Bank Control (W)
  4000245h  1  VRAMCNT_F - VRAM-F (16K) Bank Control (W)
  4000246h  1  VRAMCNT_G - VRAM-G (16K) Bank Control (W)
  4000247h  1  WRAMCNT   - WRAM Bank Control (W)
  4000248h  1  VRAMCNT_H - VRAM-H (32K) Bank Control (W)
  4000249h  1  VRAMCNT_I - VRAM-I (16K) Bank Control (W)
```

ARM9 Maths

```
  4000280h  2  DIVCNT - Division Control (R/W)
  4000290h  8  DIV_NUMER - Division Numerator (R/W)
  4000298h  8  DIV_DENOM - Division Denominator (R/W)
  40002A0h  8  DIV_RESULT - Division Quotient (=Numer/Denom) (R)
  40002A8h  8  DIVREM_RESULT - Division Remainder (=Numer MOD Denom) (R)
  40002B0h  2  SQRTCNT - Square Root Control (R/W)
  40002B4h  4  SQRT_RESULT - Square Root Result (R)
  40002B8h  8  SQRT_PARAM - Square Root Parameter Input (R/W)
  4000300h  4  POSTFLG - Undoc
  4000304h  2  POWCNT1 - Graphics Power Control Register (R/W)
```

ARM9 3D Display Engine

```
  4000320h..6A3h
```

[DS 3D I/O Map]

ARM9 Display Engine B

```
  4001000h  4    2D Engine B - DISPCNT - LCD Control (Read/Write)
  4001008h  50h  2D Engine B (same registers as GBA, some changed bits)
  400106Ch  2    2D Engine B - MASTER_BRIGHT - 16bit - Brightness Up/Down
```

ARM9 DSi Extra Registers

```
  40021Axh  ..  DSi Registers
  4004xxxh  ..  DSi Registers
```

ARM9 IPC/ROM

```
  4100000h  4    IPCFIFORECV - IPC Receive Fifo (R)
  4100010h  4    Gamecard bus 4-byte data in, for manual or dma read
```

ARM9 DS Debug Registers (Emulator/Devkits)

```
  4FFF0xxh  ..   Ensata Emulator Debug Registers
  4FFFAxxh  ..   No$gba Emulator Debug Registers
```

ARM9 Hardcoded RAM Addresses for Exception Handling

```
  27FFD9Ch   ..  NDS9 Debug Stacktop / Debug Vector (0=None)
  DTCM+3FF8h 4   NDS9 IRQ Check Bits (hardcoded RAM address)
  DTCM+3FFCh 4   NDS9 IRQ Handler (hardcoded RAM address)
```

Main Memory Control

```
  27FFFFEh  2    Main Memory Control
```

Further Memory Control Registers

[ARM CP15 System Control Coprocessor]

ARM7 I/O Map

```
  4000004h  2   DISPSTAT
  4000006h  2   VCOUNT
  40000B0h  30h DMA Channels 0..3
  4000100h  10h Timers 0..3
  4000120h  4   Debug SIODATA32
  4000128h  4   Debug SIOCNT
  4000130h  2   keyinput
  4000132h  2   keycnt
  4000134h  2   Debug RCNT
  4000136h  2   EXTKEYIN
  4000138h  1   RTC Realtime Clock Bus
  4000180h  2   IPCSYNC - IPC Synchronize Register (R/W)
  4000184h  2   IPCFIFOCNT - IPC Fifo Control Register (R/W)
  4000188h  4   IPCFIFOSEND - IPC Send Fifo (W)
  40001A0h  2   AUXSPICNT - Gamecard ROM and SPI Control
```

```
  40001A2h  2   AUXSPIDATA - Gamecard SPI Bus Data/Strobe
  40001A4h  4   Gamecard bus timing/control
  40001A8h  8   Gamecard bus 8-byte command out
  40001B0h  4   Gamecard Encryption Seed 0 Lower 32bit
  40001B4h  4   Gamecard Encryption Seed 1 Lower 32bit
  40001B8h  2   Gamecard Encryption Seed 0 Upper 7bit (bit7-15 unused)
  40001BAh  2   Gamecard Encryption Seed 1 Upper 7bit (bit7-15 unused)
  40001C0h  2   SPI bus Control (Firmware, Touchscreen, Powerman)
  40001C2h  2   SPI bus Data
```

ARM7 Memory and IRQ Control

```
  4000204h  2   EXMEMSTAT - External Memory Status
  4000206h  2   WIFIWAITCNT
  4000208h  4   IME - Interrupt Master Enable (R/W)
  4000210h  4   IE  - Interrupt Enable (R/W)
  4000214h  4   IF  - Interrupt Request Flags (R/W)
  4000218h  -   IE2  ;\DSi only (additional ARM7 interrupt sources)
  400021Ch  -   IF2  ;/
  4000240h  1   VRAMSTAT - VRAM-C,D Bank Status (R)
  4000241h  1   WRAMSTAT - WRAM Bank Status (R)
  4000300h  1   POSTFLG
  4000301h  1   HALTCNT (different bits than on GBA) (plus NOP delay)
  4000304h  2   POWCNT2  Sound/Wifi Power Control Register (R/W)
  4000308h  4   BIOSPROT - Bios-data-read-protection address
```

ARM7 Sound Registers

```
  4000400h 100h Sound Channel 0..15 (10h bytes each)
  40004x0h  4   SOUNDxCNT - Sound Channel X Control Register (R/W)
  40004x4h  4   SOUNDxSAD - Sound Channel X Data Source Register (W)
  40004x8h  2   SOUNDxTMR - Sound Channel X Timer Register (W)
  40004xAh  2   SOUNDxPNT - Sound Channel X Loopstart Register (W)
  40004xCh  4   SOUNDxLEN - Sound Channel X Length Register (W)
  4000500h  2   SOUNDCNT - Sound Control Register (R/W)
  4000504h  2   SOUNDBIAS - Sound Bias Register (R/W)
  4000508h  1   SNDCAP0CNT - Sound Capture 0 Control Register (R/W)
  4000509h  1   SNDCAP1CNT - Sound Capture 1 Control Register (R/W)
  4000510h  4   SNDCAP0DAD - Sound Capture 0 Destination Address (R/W)
  4000514h  2   SNDCAP0LEN - Sound Capture 0 Length (W)
  4000518h  4   SNDCAP1DAD - Sound Capture 1 Destination Address (R/W)
  400051Ch  2   SNDCAP1LEN - Sound Capture 1 Length (W)
```

ARM7 DSi Extra Registers

```
  40021Axh  ..  DSi Registers
  4004xxxh  ..  DSi Registers
  4004700h  2   DSi SNDEXCNT Register  ;\mapped even in DS mode
  4004C0xh  ..  DSi GPIO Registers     ;/
```

ARM7 IPC/ROM

```
  4100000h  4   IPCFIFORECV - IPC Receive Fifo (R)
  4100010h  4   Gamecard bus 4-byte data in, for manual or dma read
```

ARM7 3DS

```
  4700000h  4   Disable ARM7 bootrom overlay (W) (3DS only)
```

ARM7 WLAN Registers

```
  4800000h  ..  Wifi WS0 Region (32K) (Wifi Ports, and 8K Wifi RAM)
  4808000h  ..  Wifi WS1 Region (32K) (mirror of above, other waitstates)
```

ARM7 Hardcoded RAM Addresses for Exception Handling

```
  380FFC0h  4   DSi7 IRQ IF2 Check Bits (hardcoded RAM address) (DSi only)
  380FFDCh  ..  NDS7 Debug Stacktop / Debug Vector (0=None)
  380FFF8h  4   NDS7 IRQ IF Check Bits (hardcoded RAM address)
  380FFFCh  4   NDS7 IRQ Handler (hardcoded RAM address)
```

DS Memory Maps

NDS9 Memory Map

```
  00000000h  Instruction TCM (32KB) (not moveable) (mirror-able to 1000000h)
```

```
0xxxx000h  Data TCM        (16KB) (moveable)
02000000h  Main Memory     (4MB)
03000000h  Shared WRAM     (0KB, 16KB, or 32KB can be allocated to ARM9)
04000000h  ARM9-I/O Ports
05000000h  Standard Palettes (2KB) (Engine A BG/OBJ, Engine B BG/OBJ)
06000000h  VRAM - Engine A, BG VRAM  (max 512KB)
06200000h  VRAM - Engine B, BG VRAM  (max 128KB)
06400000h  VRAM - Engine A, OBJ VRAM (max 256KB)
06600000h  VRAM - Engine B, OBJ VRAM (max 128KB)
06800000h  VRAM - "LCDC"-allocated (max 656KB)
07000000h  OAM (2KB) (Engine A, Engine B)
08000000h  GBA Slot ROM (max 32MB)
0A000000h  GBA Slot RAM (max 64KB)
FFFF0000h  ARM9-BIOS (32KB) (only 3K used)
```
The ARM9 Exception Vectors are located at FFFF0000h. The IRQ handler redirects to [DTCM+3FFCh].

NDS7 Memory Map
```
00000000h  ARM7-BIOS (16KB)
02000000h  Main Memory (4MB)
03000000h  Shared WRAM (0KB, 16KB, or 32KB can be allocated to ARM7)
03800000h  ARM7-WRAM (64KB)
04000000h  ARM7-I/O Ports
04800000h  Wireless Communications Wait State 0 (8KB RAM at 4804000h)
04808000h  Wireless Communications Wait State 1 (I/O Ports at 4808000h)
06000000h  VRAM allocated as Work RAM to ARM7 (max 256K)
08000000h  GBA Slot ROM (max 32MB)
0A000000h  GBA Slot RAM (max 64KB)
```
The ARM7 Exception Vectors are located at 00000000h. The IRQ handler redirects to [3FFFFFCh aka 380FFFCh].

Further Memory (not mapped to ARM9/ARM7 bus)
```
3D Engine Polygon RAM (52KBx2)
3D Engine Vertex RAM (72KBx2)
Firmware (256KB) (built-in serial flash memory)
GBA-BIOS (16KB) (not used in NDS mode)
NDS Slot ROM (serial 8bit-bus, max 4GB with default protocol)
NDS Slot FLASH/EEPROM/FRAM (serial 1bit-bus)
```

Shared-RAM
Even though Shared WRAM begins at 3000000h, programs are commonly using mirrors at 37F8000h (both ARM9 and ARM7). At the ARM7-side, this allows to use 32K Shared WRAM and 64K ARM7-WRAM as a continous 96K RAM block.

Undefined I/O Ports
On the NDS (at the ARM9-side at least) undefined I/O ports are always zero.

Undefined Memory Regions
16MB blocks that do not contain any defined memory regions (or that contain only mapped TCM regions) are typically completely undefined.
16MB blocks that do contain valid memory regions are typically containing mirrors of that memory in the unused upper part of the 16MB area (only exceptions are TCM and BIOS which are not mirrored).

DS Memory Control

Memory Control
DS Memory Control - Cache and TCM
DS Memory Control - Cartridges and Main RAM

DS Memory Control - WRAM
DS Memory Control - VRAM
DS Memory Control - BIOS

Memory Access Time
DS Memory Timings

DS Memory Control - Cache and TCM

TCM and Cache are controlled by the System Control Coprocessor,
ARM CP15 System Control Coprocessor

The specifications for the NDS9 are:

Tightly Coupled Memory (TCM)
```
ITCM 32K, base=00000000h (fixed, not move-able)
DTCM 16K, base=moveable  (default base=27C0000h)
```
Note: Although ITCM is NOT moveable, the NDS Firmware configures the ITCM size to 32MB, and so, produces ITCM mirrors at 0..1FFFFFFh. Furthermore, the PU can be used to lock/unlock memory in that region. That trick allows to move ITCM anywhere within the lower 32MB of memory.

Cache
```
Data Cache 4KB, Instruction Cache 8KB
4-way set associative method
Cache line 8 words (32 bytes)
Read-allocate method (ie. writes are not allocating cache lines)
Round-robin and Pseudo-random replacement algorithms selectable
Cache Lockdown, Instruction Prefetch, Data Preload
Data write-through and write-back modes selectable
```

Protection Unit (PU)
Recommended/default settings are:

Region	Name	Address	Size	Cache	WBuf	Code	Data
-	Background	00000000h	4GB	-	-	-	-
0	I/O and VRAM	04000000h	64MB	-	-	R/W	R/W
1	Main Memory	02000000h	4MB	On	On	R/W	R/W
2	ARM7-dedicated	027C0000h	256KB	-	-	-	-
3	GBA Slot	08000000h	128MB	-	-	-	R/W
4	DTCM	027C0000h	16KB	-	-	-	R/W
5	ITCM	01000000h	32KB	-	-	R/W	R/W
6	BIOS	FFFF0000h	32KB	On	-	R	R
7	Shared Work	027FF000h	4KB	-	-	-	R/W

Notes: In Nintendo's hardware-debugger, Main Memory is expanded to 8MB (for that reason, some addresses are at 27NN000h instead 23NN000h) (some of the extra memory is reserved for the debugger, some can be used for game development). Region 2 and 7 are not understood? GBA Slot should be max 32MB+64KB, rounded up to 64MB, no idea why it is 128MB? DTCM and ITCM do not use Cache and Write-Buffer because TCM is fast. Above settings do not allow to access Shared Memory at 37F8000h? Do not use cache/wbuf for I/O, doing so might suppress writes, and/or might read outdated values.
The main purpose of the Protection Unit is debugging, a major problem with GBA programs have been faulty accesses to memory address 00000000h and up (due to [base+offset] addressing with uninitialized (zero) base values). This problem has been fixed in the NDS, for the ARM9 processor at least, still there are various leaks: For example, the 64MB I/O and VRAM area contains only ca. 660KB valid addresses, and the ARM7 probably doesn't have a Protection Unit at all. Alltogether, the protection is better than in GBA, but it's still pretty crude compared with software debugging tools.
Region address/size are unified (same for code and data), however, cachabilty and access rights are non-unified (and may be separately defined for code and data).

Note: The NDS7 doesn't have any TCM, Cache, or CP15.

DS Memory Control - Cartridges and Main RAM

4000204h - NDS9 - EXMEMCNT - 16bit - External Memory Control (R/W)
4000204h - NDS7 - EXMEMSTAT - 16bit - External Memory Status (R/W..R)

```
0-1   32-pin GBA Slot SRAM Access Time    (0-3 = 10, 8, 6, 18 cycles)
2-3   32-pin GBA Slot ROM 1st Access Time (0-3 = 10, 8, 6, 18 cycles)
4     32-pin GBA Slot ROM 2nd Access Time (0-1 = 6, 4 cycles)
5-6   32-pin GBA Slot PHI-pin out   (0-3 = Low, 4.19MHz, 8.38MHz, 16.76MHz)
7     32-pin GBA Slot Access Rights     (0=ARM9, 1=ARM7)
8-10  Not used (always zero)
11    17-pin NDS Slot Access Rights     (0=ARM9, 1=ARM7)
12    Not used (always zero)
13    NDS:Always set?  ;set/tested by DSi bootcode: Main RAM enable, CE2 pin?
14    Main Memory Interface Mode Switch (0=Async/GBA/Reserved, 1=Synchronous)
15    Main Memory Access Priority       (0=ARM9 Priority, 1=ARM7 Priority)
```

Bit0-6 can be changed by both NDS9 and NDS7, changing these bits affects the local EXMEM register only, not that of the other CPU.
Bit7-15 can be changed by NDS9 only, changing these bits affects both EXMEM registers, ie. both NDS9 and NDS7 can read the current NDS9 setting.
Bit14=0 is intended for GBA mode, however, writes to this bit appear to be ignored?
DS Main Memory Control

GBA Slot (8000000h-AFFFFFFh)
The GBA Slot can be mapped to ARM9 or ARM7 via EXMEMCNT.7.
For the selected CPU, memory at 8000000h-9FFFFFFh contains the "GBA ROM" region, and memory at A000000h-AFFFFFFh contains the "GBA SRAM" region (repeated every 64Kbytes). If there is no cartridge in GBA Slot, then the ROM/SRAM regions will contain open-bus values: SRAM region is FFh-filled (High-Z). And ROM region is filled by increasing 16bit values (Addr/2), possibly ORed with garbage depending on the selected ROM Access Time:

```
6 clks   --> returns "Addr/2"
8 clks   --> returns "Addr/2"
10 clks  --> returns "Addr/2 OR FE08h" (or similar garbage)
18 clks  --> returns "FFFFh" (High-Z)
```

For the deselected CPU, all memory at 8000000h-AFFFFFFh becomes 00h-filled, this is required for bugged games like Digimon Story: Super Xros Wars (which is accidently reading deselected GBA SRAM at [main_ram_base+main_ram_addr*4], whereas it does presumably want to read Main RAM at [main_ram_base+index*4]).

DS Memory Control - WRAM

4000247h - NDS9 - WRAMCNT - 8bit - WRAM Bank Control (R/W)
4000241h - NDS7 - WRAMSTAT - 8bit - WRAM Bank Status (R)
Should not be changed when using Nintendo's API.

```
0-1   ARM9/ARM7 (0-3 = 32K/0K, 2nd 16K/1st 16K, 1st 16K/2nd 16K, 0K/32K)
2-7   Not used
```

The ARM9 WRAM area is 3000000h-3FFFFFFh (16MB range).
The ARM7 WRAM area is 3000000h-37FFFFFh (8MB range).
The allocated 16K or 32K are mirrored everywhere in the above areas.
De-allocation (0K) is a special case: At the ARM9-side, the WRAM area is then empty (containing undefined data). At the ARM7-side, the WRAM area is then containing mirrors of the 64KB ARM7-WRAM (the memory at 3800000h and up).

DS Memory Control - VRAM

4000240h - NDS7 - VRAMSTAT - 8bit - VRAM Bank Status (R)

```
0     VRAM C enabled and allocated to NDS7  (0=No, 1=Yes)
1     VRAM D enabled and allocated to NDS7  (0=No, 1=Yes)
2-7   Not used (always zero)
```

The register indicates if VRAM C/D are allocated to NDS7 (as Work RAM), ie. if VRAMCNT_C/D are enabled (Bit7=1), with MST=2 (Bit0-2). However, it does not reflect the OFS value.

4000240h - NDS9 - VRAMCNT_A - 8bit - VRAM-A (128K) Bank Control (W)
4000241h - NDS9 - VRAMCNT_B - 8bit - VRAM-B (128K) Bank Control (W)
4000242h - NDS9 - VRAMCNT_C - 8bit - VRAM-C (128K) Bank Control (W)
4000243h - NDS9 - VRAMCNT_D - 8bit - VRAM-D (128K) Bank Control (W)
4000244h - NDS9 - VRAMCNT_E - 8bit - VRAM-E (64K) Bank Control (W)
4000245h - NDS9 - VRAMCNT_F - 8bit - VRAM-F (16K) Bank Control (W)
4000246h - NDS9 - VRAMCNT_G - 8bit - VRAM-G (16K) Bank Control (W)
4000248h - NDS9 - VRAMCNT_H - 8bit - VRAM-H (32K) Bank Control (W)
4000249h - NDS9 - VRAMCNT_I - 8bit - VRAM-I (16K) Bank Control (W)

```
0-2   VRAM MST              ;Bit2 not used by VRAM-A,B,H,I
3-4   VRAM Offset (0-3)     ;Offset not used by VRAM-E,H,I
5-6   Not used
7     VRAM Enable (0=Disable, 1=Enable)
```

There is a total of 656KB of VRAM in Blocks A-I.
Table below shows the possible configurations.

VRAM	SIZE	MST	OFS	ARM9, Plain ARM9-CPU Access (so-called LCDC mode)
A	128K	0	-	6800000h-681FFFFh
B	128K	0	-	6820000h-683FFFFh
C	128K	0	-	6840000h-685FFFFh
D	128K	0	-	6860000h-687FFFFh
E	64K	0	-	6880000h-688FFFFh
F	16K	0	-	6890000h-6893FFFh
G	16K	0	-	6894000h-6897FFFh
H	32K	0	-	6898000h-689FFFFh
I	16K	0	-	68A0000h-68A3FFFh
VRAM	**SIZE**	**MST**	**OFS**	**ARM9, 2D Graphics Engine A, BG-VRAM (max 512K)**
A,B,C,D	128K	1	0..3	6000000h+(20000h*OFS)
E	64K	1	-	6000000h
F,G	16K	1	0..3	6000000h+(4000h*OFS.0)+(10000h*OFS.1)
VRAM	**SIZE**	**MST**	**OFS**	**ARM9, 2D Graphics Engine A, OBJ-VRAM (max 256K)**
A,B	128K	2	0..1	6400000h+(20000h*OFS.0) ;(OFS.1 must be zero)
E	64K	2	-	6400000h
F,G	16K	2	0..3	6400000h+(4000h*OFS.0)+(10000h*OFS.1)
VRAM	**SIZE**	**MST**	**OFS**	**2D Graphics Engine A, BG Extended Palette**
E	64K	4	-	Slot 0-3 ;only lower 32K used
F,G	16K	4	0..1	Slot 0-1 (OFS=0), Slot 2-3 (OFS=1)
VRAM	**SIZE**	**MST**	**OFS**	**2D Graphics Engine A, OBJ Extended Palette**
F,G	16K	5	-	Slot 0 ;16K each (only lower 8K used)
VRAM	**SIZE**	**MST**	**OFS**	**Texture/Rear-plane Image**
A,B,C,D	128K	3	0..3	Slot OFS(0-3) ;(Slot2-3: Texture, or Rear-plane)
VRAM	**SIZE**	**MST**	**OFS**	**Texture Palette**
E	64K	3	-	Slots 0-3 ;OFS=don't care
F,G	16K	3	0..3	Slot (OFS.0*1)+(OFS.1*4) ;ie. Slot 0, 1, 4, or 5
VRAM	**SIZE**	**MST**	**OFS**	**ARM9, 2D Graphics Engine B, BG-VRAM (max 128K)**
C	128K	4	-	6200000h
H	32K	1	-	6200000h
I	16K	1	-	6208000h
VRAM	**SIZE**	**MST**	**OFS**	**ARM9, 2D Graphics Engine B, OBJ-VRAM (max 128K)**
D	128K	4	-	6600000h

```
I       16K   2    -     6600000h
VRAM    SIZE  MST  OFS   2D Graphics Engine B, BG Extended Palette
H       32K   2    -     Slot 0-3
VRAM    SIZE  MST  OFS   2D Graphics Engine B, OBJ Extended Palette
I       16K   3    -     Slot 0  ;(only lower 8K used)
VRAM    SIZE  MST  OFS   <ARM7>, Plain <ARM7>-CPU Access
C,D     128K  2    0..1  6000000h+(20000h*OFS.0)  ;OFS.1 must be zero
```

Notes
In Plain-CPU modes, VRAM can be accessed only by the CPU (and by the Capture Unit, and by VRAM Display mode). In "Plain <ARM7>-CPU Access" mode, the VRAM blocks are allocated as Work RAM to the NDS7 CPU.
In BG/OBJ VRAM modes, VRAM can be accessed by the CPU at specified addresses, and by the display controller.
In Extended Palette and Texture Image/Palette modes, VRAM is not mapped to CPU address space, and can be accessed only by the display controller (so, to initialize or change the memory, it should be temporarily switched to Plain-CPU mode).
All VRAM (and Palette, and OAM) can be written to only in 16bit and 32bit units (STRH, STR opcodes), 8bit writes are ignored (by STRB opcode). The only exception is "Plain <ARM7>-CPU Access" mode: The ARM7 CPU can use STRB to write to VRAM (the reason for this special feature is that, in GBA mode, two 128K VRAM blocks are used to emulate the GBA's 256K Work RAM).

Other Video RAM
Aside from the map-able VRAM blocks, there are also some video-related memory regions at fixed addresses:

```
5000000h Engine A Standard BG Palette (512 bytes)
5000200h Engine A Standard OBJ Palette (512 bytes)
5000400h Engine B Standard BG Palette (512 bytes)
5000600h Engine B Standard OBJ Palette (512 bytes)
7000000h Engine A OAM (1024 bytes)
7000400h Engine B OAM (1024 bytes)
```

DS Memory Control - BIOS

4000308h - NDS7 - BIOSPROT - Bios-data-read-protection address
Used to double-protect the first some KBytes of the NDS7 BIOS. The BIOS is split into two protection regions, one always active, one controlled by the BIOSPROT register. The overall idea is that only the BIOS can read from itself, any other attempts to read from that regions return FFh-bytes.

```
Opcodes at...      Can read from      Expl.
0..[BIOSPROT]-1    0..3FFFh           Double-protected (when BIOSPROT is set)
[BIOSPROT]..3FFFh  [BIOSPROT]..3FFFh  Normal-protected (always active)
```

The initial BIOSPROT setting on power-up is zero (disabled). Before starting the cartridge, the BIOS boot code sets the register to 1204h (actually 1205h, but the mis-aligned low-bit is ignored). Once when initialized, further writes to the register are ignored.

The double-protected region contains the exception vectors, some bytes of code, and the cartridge KEY1 encryption seed (about 4KBytes). As far as I know, it is impossible to unlock the memory once when it is locked, however, with some trickery, it is possible execute code before it gets locked. Also, the two THUMB opcodes at 05ECh can be used to read all memory at 0..3FFFh,

```
05ECh  ldrb r3,[r3,12h]      ;requires incoming r3=src-12h
05EEh  pop  r2,r4,r6,r7,r15  ;requires dummy values & THUMB retadr on stack
```

Additionally most BIOS functions (eg. CpuSet), include a software-based protection which rejects source addresses in the BIOS area (the only exception is GetCRC16, though it still cannot bypass the BIOSPROT setting).

Note

The NDS9 BIOS doesn't include any software or hardware based read protection.

DS Memory Timings

System Clock

```
Bus clock  = 33MHz (33.513982 MHz) (1FF61FEh Hertz)
NDS7 clock = 33MHz (same as bus clock)
NDS9 clock = 66MHz (internally twice bus clock; for cache/tcm)
```

Most timings in this document are specified for 33MHz clock (not for the 66MHz clock). Respectively, NDS9 timings are counted in "half" cycles.

Memory Access Times

Tables below show the different access times for code/data fetches on arm7/arm9 cpus, measured for sequential/nonsequential 32bit/16bit accesses.

```
NDS7/CODE             NDS9/CODE
N32 S32 N16 S16 Bus   N32 S32 N16 S16 Bus
9   2   8   1   16    9   9   4.5 4.5 16  Main RAM (read) (cache off)
1   1   1   1   32    4   4   2   2   32  WRAM,BIOS,I/O,OAM
2   2   1   1   16    5   5   2.5 2.5 16  VRAM,Palette RAM
16  12  10  6   16    19  19  9.5 9.5 16  GBA ROM (example 10,6 access)
-   -   -   -   -     0.5 0.5 0.5 0.5 32  TCM, Cache_Hit
-   -   -   -   -     (--Load 8 words--)  Cache_Miss

NDS7/DATA             NDS9/DATA
N32 S32 N16 S16 Bus   N32 S32 N16 S16 Bus
10  2   9   1   16    10  2   9   1   16  Main RAM (read) (cache off)
1   1   1   1   32    4   1   4   1   32  WRAM,BIOS,I/O,OAM
1?  2   1   1   16    5   2   4   1   16  VRAM,Palette RAM
15  12  9   6   16    19  12  13  6   16  GBA ROM (example 10,6 access)
9   10  9   10  8     13  10  13  10  8   GBA RAM (example 10 access)
-   -   -   -   -     0.5 0.5 0.5 -   32  TCM, Cache_Hit
-   -   -   -   -     (--Load 8 words--)  Cache_Miss
-   -   -   -   -     11  11  11  -   32  Cache_Miss (BIOS)
-   -   -   -   -     23  23  23  -   16  Cache_Miss (Main RAM)
```

All timings are counted in 33MHz units (so "half" cycles can occur on NDS9).
Note: 8bit data accesses have same timings than 16bit data.

*** DS Memory Timing Notes ***

The NDS timings are altogether pretty messed up, with different timings for CODE and DATA fetches, and different timings for NDS7 and NDS9...

NDS7/CODE

Timings for this region can be considered as "should be" timings.

NDS7/DATA

Quite the same as NDS7/CODE. Except that, nonsequential Main RAM accesses are 1 cycle slower, and more strange, nonsequential GBA Slot accesses are 1 cycle faster.

NDS9/CODE

This is the most messiest timing. An infamous PENALTY of 3 cycles is added to all nonsequential accesses (except cache, tcm, and main ram). And, all opcode fetches are forcefully made nonsequential 32bit (the NDS9 simply doesn't support fast sequential opcode fetches). That applies also for THUMB code (two 16bit opcodes are fetched by a single nonsequential 32bit access) (so the time per 16bit opcode is one half of the 32bit fetch) (unless a branch causes only one of the two 16bit opcodes to be executed, then that opcode will have the full 32bit access time).

NDS9/DATA
Allows both sequential and nonsequential access, and both 16bit and 32bit access, so it's faster than NDS9/CODE. Nethertheless, it's still having the 3 cycle PENALTY on nonsequential accesses. And, similar as NDS7/DATA, it's also adding 1 cycle to nonsequential Main RAM accesses.

*** More Timing Notes / Lots of unsorted Info ***

Actual CPU Performance
The 33MHz NDS7 is running more or less nicely at 33MHz. However, the so-called "66MHz" NDS9 is having <much> higher waitstates, and it's effective bus speed is barely about 8..16MHz, the only exception is code/data in cache/tcm, which is eventually reaching real 66MHz (that, assuming cache HITS, otherwise, in case of cache MISSES, the cached memory timing might even drop to 1.4MHz or so?).
****
ARM9 opcode fetches are always N32 + 3 waits.
```
  S16 and N16 do not exist (because thumb-double-fetching) (see there).
  S32 becomes N32 (ie. the ARM9 does NOT support fast sequential timing).
```
That N32 is having same timing as normal N32 access on NDS7, plus 3 waits.
```
  Eg. an ARM9 N32 or S32 to 16bit bus will take: N16 + S16 + 3 waits.
  Eg. an ARM9 N32 or S32 to 32bit bus will take: N32 + 3 waits.
```
Main Memory is ALWAYS having the nonsequential 3 wait PENALTY (even on ARM7).
****
ARM9 Data fetches however are allowed to use sequential timing, as well as raw 16bit accesses (which aren't forcefully expanded to slow 32bit accesses).
Nethertheless, the 3 wait PENALTY is added to any NONSEQUENTIAL accesses.
Only exceptions are cache and tcm which do not have that penalty.
```
 Eg. LDRH on 16bit-data-bus is N16+3waits.
 Eg. LDR  on 16bit-data-bus is N16+S16+3waits.
 Eg. LDM  on 16bit-data-bus is N16+(n*2-1)*S16+3waits.
```
Eventually, data fetches can take place parallel with opcode fetches.
```
 That is NOT true for LDM (works only for LDR/LDRB/LDRH).
 That is NOT true for DATA in SAME memory region than CODE.
 That is NOT true for DATA in ITCM (no matter if CODE is in ITCM).
**********
```

NDS9 Busses
Unlike ARM7, the ARM9 has separate code and data busses, allowing it to perform code and data fetches simultaneously (provided that both are in different memory regions).
Normally, opcode execution times are calculated as "(codetime+datatime)", with the two busses, it can (ideally) be "MAX(codetime,datatime)", so the data access time may virtually take "NULL" clock cycles.
In practice, DTCM and Data Cache access can take NULL cycles (however, data access to ITCM can't).
When executing code in cache/itcm, data access to non-cache/tcm won't be any faster than with only one bus (as it's best, it could subtract 0.5 cycles from datatime, but, the access must be "aligned" to the bus-clock, so the "datatime-0.5" will be rounded back to the original "datatime").
When executing code in uncached main ram, and accessing data (elsewhere than in main memory, cache/tcm), then execution time is typically "codetime+datatime-2".

NDS9 Internal Cycles
Additionally to codetime+datatime, some opcodes include one or more internal cycles. Compared with ARM7, the behaviour of that internal cycles is slightly different on ARM9. First of, on the NDS9, the internal cycles are of course "half" cycles (ie. counted in 66MHz units, not in 33MHz units) (although they may get rounded to "full" cycles upon next memory access outside tcm/cache). And, the ARM9 is in some cases "skipping" the internal cycles, that often depending on whether or not the next opcode is using the result of the current opcode. Another big difference is that the ARM9 has lost the fast-multiply feature for small numbers; in some cases that may result in faster execution, but may also result in slower execution (one workaround would be to manually replace MUL opcodes by the new ARM9 halfword multiply opcodes); the slowest case are MUL opcodes that

do update flags (eg. MULS, MLAS, SMULLS, etc. in ARM mode, and all ALL multiply opcodes in THUMB mode).

NDS9 Thumb Code
In thumb mode, the NDS9 is fetching two 16bit opcodes by a single 32bit read. In case of 32bit bus, this reduces the amount of memory traffic and may result in faster execution time, of course that works only if the two opcodes are within a word-aligned region (eg. loops at word-aligned addresses will be faster than non-aligned loops). However, the double-opcode-fetching is also done on 16bit bus memory, including for unnecessary fetches, such like opcodes after branch commands, so the feature may cause heavy slowdowns.

Main Memory
Reportedly, the main memory access times would be 5 cycles (nonsequential read), 4 cycles (nonsequential write), and 1 cycle (sequential read or write). Plus whatever termination cycles. Plus 3 cycles on nonsequential access to the last 2-bytes of a 32-byte block.
That's of course all wrong. Reads are much slower than 5 cycles. Not yet tested if writes are faster. And, I haven't been able to reproduce the 3 cycles on last 2-bytes effect, actually, it looks more as if that 3 cycles are accidently added to ALL nonsequential accesses, at ALL main memory addresses, and even to most OTHER memory regions... which might be the source of the PENALTY which occurs on VRAM/WRAM/OAM/Palette and I/O accesses.

DMA
In some cases DMA main memory read cycles are reportedly performed simultaneously with DMA write cycles to other memory.

NDS9
On the NDS9, all external memory access (and I/O) is delayed to bus clock (or actually MUCH slower due to the massive waitstates), so the full 66MHz can be used only internally in the NDS9 CPU core, ie. with cache and TCM.

Bus Clock
The exact bus clock is specified as 33.513982 MHz (1FF61FEh Hertz). However, on my own NDS, measured in relation to the RTC seconds IRQ, it appears more like 1FF6231h, that inaccuary of 1 cycle per 657138 cycles (about one second per week) on either oscillator, isn't too significant though.

GBA Slot
The access time for GBA slot can be configured via EXMEMCNT register.

VRAM Waitstates
Additionally, on NDS9, a one cycle wait can be added to VRAM accesses (when the video controller simultaneously accesses it) (that can be disabled by Forced Blank, see DISPCNT.Bit7). Moreover, additional VRAM waitstates occur when using the video capture function.
Note: VRAM being mapped to NDS7 is always free of additional waits.

DS Video

The NDS has two 2D Video Engines, each basically the same as in GBA, see
GBA LCD Video Controller

NDS Specific 2D Video Features
DS Video Stuff
DS Video BG Modes / Control
DS Video OBJs

DS Video Extended Palettes
DS Video Capture and Main Memory Display Mode
DS Video Display System Block Diagram

For Display Power Control (and Display Swap), and VRAM Allocation, see
DS Power Control
DS Power Management Device
DS Memory Control - VRAM

DS Video Stuff

DS Display Dimensions / Timings
Dot clock = 5.585664 MHz (=33.513982 MHz / 6)
H-Timing: 256 dots visible, 99 dots blanking, 355 dots total (15.7343KHz)
V-Timing: 192 lines visible, 71 lines blanking, 263 lines total (59.8261 Hz)
The V-Blank cycle for the 3D Engine consists of the 23 lines, 191..213.
Screen size 62.5mm x 47.0mm (each) (256x192 pixels)
Vertical space between screens 22mm (equivalent to 90 pixels)

400006Ch - NDS9 - MASTER_BRIGHT - 16bit - Master Brightness Up/Down
```
0-4   Factor used for 6bit R,G,B Intensities (0-16, values >16 same as 16)
        Brightness up:   New = Old + (63-Old) * Factor/16
        Brightness down: New = Old - Old      * Factor/16
5-13  Not used
14-15 Mode (0=Disable, 1=Up, 2=Down, 3=Reserved)
16-31 Not used
```

DISPSTAT/VCOUNT
The LY and LYC values are in range 0..262, so LY/LYC values have been expanded to 9bit values: LY = VCOUNT Bit 0..8, and LYC=DISPSTAT Bit8..15,7.
VCOUNT register is write-able, allowing to synchronize linked DS consoles.
For proper synchronization:
```
write new LY values only in range of 202..212
write only while old LY values are in range of 202..212
```
DISPSTAT/VCOUNT supported by NDS9 (Engine A Ports, without separate Engine B Ports), and by NDS7 (allowing to synchronize NDS7 with display timings).
Similar as on GBA, the VBlank flag isn't set in the last line (ie. only in lines 192..261, but not in line 262).
Although the drawing time is only 1536 cycles (256*6), the NDS9 H-Blank flag is "0" for a total of 1606 cycles (and, for whatever reason, a bit longer, 1613 cycles in total, on NDS7).

VRAM Waitstates
The display controller performs VRAM-reads once every 6 clock cycles, a 1 cycle waitstate is generated if the CPU simultaneously accesses VRAM. With capture enabled, additionally VRAM-writes take place once every 6 cycles, so the total VRAM-read/write access rate is then once every 3 cycles.

DS Window Glitches
The DS counts scanlines in range 0..262 (0..106h), of which only the lower 8bit are compared with the WIN0V/WIN1V register settings. Respectively, Y1 coordinates 00h..06h will be triggered in scanlines 100h-106h by mistake. That means, the window gets activated within VBlank period, and will be active in scanline 0 and up (that is no problem with Y1=0, but Y1=1..6 will appear as if if Y1 would be 0). Workaround would be to disable the Window during VBlank, or to change Y1 during VBlank (to a value that does not occur during VBlank period, ie. 7..191).
Also, there's a problem to fit the 256 pixel horizontal screen resolution into 8bit values: X1=00h is treated as 0 (left-most), X2=00h is treated as 100h (right-most). However, the window is not displayed if X1=X2=00h; the

window width can be max 255 pixels.

2D Engines
Includes two 2D Engines, called A and B. Both engines are accessed by the ARM9 processor, each using different memory and register addresses:

Region	Engine A	Engine B
I/O Ports	4000000h	4001000h
Palette	5000000h (1K)	5000400h (1K)
BG VRAM	6000000h (max 512K)	6200000h (max 128K)
OBJ VRAM	6400000h (max 256K)	6600000h (max 128K)
OAM	7000000h (1K)	7000400h (1K)

Engine A additionally supports 3D and large-screen 256-color Bitmaps, plus main-memory-display and vram-display modes, plus capture unit.

Viewing Angles
The LCD screens are best viewed at viewing angles of 90 degrees. Colors may appear distorted, and may even become invisible at other viewing angles.
When the console is handheld, both screens can be turned into preferred direction. When the console is settled on a table, only the upper screen can be turned, but the lower screen is stuck into horizontal position - which results in rather bad visibility (unless the user moves his/her head directly above of it).

4000070h - NDS9 - TVOUTCNT - Unknown (W)

Bit0-3	"COMMAND" (?)
Bit4-7	"COMMAND2" (?)
Bit8-11	"COMMAND3" (?)

This register has been mentioned in an early I/O map from Nintendo, as far as I know, the register isn't used by any games/firmware/bios, not sure if it does really exist on release-version, or if it's been prototype stuff...?

DS-Lite Screens
The screens in the DS-Lite seem to allow a wider range of vertical angles.
The bad news is that the colors of the DS-Lite are (no surprise) not backwards compatible with older NDS and GBA displays. The good news is that Nintendo has finally reached near-CRT-quality (without blurred colors), so one could hope that they won't show up with more displays with other colors in future.
Don't know if there's an official/recommended way to detect DS-Lite displays (?) possible methods would be whatever values in Firmware header, or by functionality of Power Managment device, or (not too LCD-related) by Wifi Chip ID.

DS Video BG Modes / Control

4000000h - NDS9 - DISPCNT

Bit	Engine	Expl.
0-2	A+B	BG Mode
3	A	BG0 2D/3D Selection (instead CGB Mode) (0=2D, 1=3D)
4	A+B	Tile OBJ Mapping (0=2D; max 32KB, 1=1D; max 32KB..256KB)
5	A+B	Bitmap OBJ 2D-Dimension (0=128x512 dots, 1=256x256 dots)
6	A+B	Bitmap OBJ Mapping (0=2D; max 128KB, 1=1D; max 128KB..256KB)
7-15	A+B	Same as GBA
16-17	A+B	Display Mode (Engine A: 0..3, Engine B: 0..1, GBA: Green Swap)
18-19	A	VRAM block (0..3=VRAM A..D) (For Capture & above Display Mode=2)
20-21	A+B	Tile OBJ 1D-Boundary (see Bit4)
22	A	Bitmap OBJ 1D-Boundary (see Bit5-6)
23	A+B	OBJ Processing during H-Blank (was located in Bit5 on GBA)
24-26	A	Character Base (in 64K steps) (merged with 16K step in BGxCNT)
27-29	A	Screen Base (in 64K steps) (merged with 2K step in BGxCNT)
30	A+B	BG Extended Palettes (0=Disable, 1=Enable)
31	A+B	OBJ Extended Palettes (0=Disable, 1=Enable)

BG Mode

Engine A BG Mode (DISPCNT LSBs) (0-6, 7=Reserved)

Mode	BG0	BG1	BG2	BG3
0	Text/3D	Text	Text	Text
1	Text/3D	Text	Text	Affine
2	Text/3D	Text	Affine	Affine
3	Text/3D	Text	Text	Extended
4	Text/3D	Text	Affine	Extended
5	Text/3D	Text	Extended	Extended
6	3D	-	Large	-

Of which, the "Extended" modes are sub-selected by BGxCNT bits:

BGxCNT.Bit7	BGxCNT.Bit2	Extended Affine Mode Selection
0	CharBaseLsb	rot/scal with 16bit bgmap entries (Text+Affine mixup)
1	0	rot/scal 256 color bitmap
1	1	rot/scal direct color bitmap

Engine B: Same as above, except that: Mode 6 is reserved (no Large screen bitmap), and BG0 is always Text (no 3D support).

Affine = formerly Rot/Scal mode (with 8bit BG Map entries)

Large Screen Bitmap = rot/scal 256 color bitmap (using all 512K of 2D VRAM)

Display Mode (DISPCNT.16-17):

```
0  Display off (screen becomes white)
1  Graphics Display (normal BG and OBJ layers)
2  Engine A only: VRAM Display (Bitmap from block selected in DISPCNT.18-19)
3  Engine A only: Main Memory Display (Bitmap DMA transfer from Main RAM)
```

Mode 2-3 display a raw direct color bitmap (15bit RGB values, the upper bit in each halfword is unused), without any further BG,OBJ,3D layers, these modes are completely bypassing the 2D/3D engines as well as any 2D effects, however the Master Brightness effect can be applied to these modes. Mode 2 is particulary useful to display captured 2D/3D images (in that case it can indirectly use the 2D/3D engine).

BGxCNT

character base extended from bit2-3 to bit2-5 (bit4-5 formerly unused)

```
engine A screen base: BGxCNT.bits*2K + DISPCNT.bits*64K
engine B screen base: BGxCNT.bits*2K + 0
engine A char base: BGxCNT.bits*16K + DISPCNT.bits*64K
engine B char base: BGxCNT.bits*16K + 0
```

char base is used only in tile/map modes (not bitmap modes)

screen base is used in tile/map modes,

screen base used in bitmap modes as BGxCNT.bits*16K, without DISPCNT.bits*64K

screen base however NOT used at all for Large screen bitmap mode

bgcnt size	text	rotscal	bitmap	large bmp
0	256x256	128x128	128x128	512x1024
1	512x256	256x256	256x256	1024x512
2	256x512	512x512	512x256	-
3	512x512	1024x1024	512x512	-

bitmaps that require more than 128K VRAM are supported on engine A only.

For BGxCNT.Bit7 and BGxCNT.Bit2 in Extended Affine modes, see above BG Mode description (extended affine doesn't include 16-color modes, so color depth bit can be used for mode selection. Also, bitmap modes do not use charbase, so charbase.0 can be used for mode selection as well).

```
for BG0CNT, BG1CNT only: bit13 selects extended palette slot
                         (BG0: 0=Slot0, 1=Slot2, BG1: 0=Slot1, 1=Slot3)
```

Direct Color Bitmap BG, and Direct Color Bitmap OBJ

BG/OBJ Supports 32K colors (15bit RGB value) - so far same as GBAs BG.

However, the upper bit (Bit15) is used as Alpha flag. That is, Alpha=0=Transparent, Alpha=1=Normal (ie. on

the NDS, Direct Color values 0..7FFFh are NOT displayed).

Unlike GBA bitmap modes, NDS bitmap modes are supporting the Area Overflow bit (BG2CNT and BG3CNT, Bit 13).

DS Video OBJs

DS OBJ Priority
The GBA has been assigning OBJ priority in respect to the 7bit OAM entry number, regardless of the OBJs 2bit BG-priority attribute (which allowed to specify invalid priority orders). That problem has been fixed in DS mode by combining the above two values into a 9bit priority value.

OBJ Tile Mapping (DISPCNT.4,20-21):

```
Bit4  Bit20-21  Dimension Boundary Total ;Notes
0     x         2D        32       32K   ;Same as GBA 2D Mapping
1     0         1D        32       32K   ;Same as GBA 1D Mapping
1     1         1D        64       64K
1     2         1D        128      128K
1     3         1D        256      256K  ;Engine B: 128K max
```

TileVramAddress = TileNumber * BoundaryValue
Even if the boundary gets changed, OBJs are kept composed of 8x8 tiles.

Bitmap OBJ Mapping (DISPCNT.6,5,22):
Bitmap OBJs are 15bit Direct Color data, plus 1bit Alpha flag (in bit15).

```
Bit6 Bit5 Bit22 Dimension    Boundary   Total ;Notes
0    0    x     2D/128 dots  8x8 dots   128K  ;Source Bitmap width 128 dots
0    1    x     2D/256 dots  8x8 dots   128K  ;Source Bitmap width 256 dots
1    0    0     1D           128 bytes  128K  ;Source Width = Target Width
1    0    1     1D           256 bytes  256K  ;Engine A only
1    1    x     Reserved
```

In 1D mapping mode, the Tile Number is simply multiplied by the boundary value.

```
1D_BitmapVramAddress = TileNumber(0..3FFh) * BoundaryValue(128..256)
2D_BitmapVramAddress = (TileNo AND MaskX)*10h + (TileNo AND NOT MaskX)*80h
```

In 2D mode, the Tile Number is split into X and Y indices, the X index is located in the LSBs (ie. MaskX=0Fh, or MaskX=1Fh, depending on DISPCNT.5).

OBJ Attribute 0 and 2
Setting the OBJ Mode bits (Attr 0, Bit10-11) to a value of 3 has been prohibited in GBA, however, in NDS it selects the new Bitmap OBJ mode; in that mode, the Color depth bit (Attr 0, Bit13) should be set to zero; also in that mode, the color bits (Attr 2, Bit 12-15) are used as Alpha-OAM value (instead of as palette setting).

OBJ Vertical Wrap
On the GBA, a large OBJ (with 64pix height, scaled into double-size region of 128pix height) located near the bottom of the screen has been wrapped to the top of the screen (and was NOT displayed at the bottom of the screen).
This problem has been "corrected" in the NDS (except in GBA mode), that is, on the NDS, the OBJ appears BOTH at the top and bottom of the screen. That isn't necessarily better - the advantage is that one can manually enable/disable the OBJ in the desired screen-half on IRQ level; that'd be required only if the wrapped portion is non-transparent.

DS Video Extended Palettes

Extended Palettes

When allocating extended palettes, the allocated memory is not mapped to the CPU bus, so the CPU can access extended palette only when temporarily de-allocating it.

Color 0 of all standard/extended palettes is transparent, color 0 of BG standard palette 0 is used as backdrop. extended palette memory must be allocated to VRAM.

BG Extended Palette enabled in DISPCNT Bit 30, when enabled,

```
 standard palette --> 16-color tiles (with 16bit bgmap entries) (text)
                      256-color tiles (with 8bit bgmap entries) (rot/scal)
                      256-color bitmaps
                      backdrop-color (color 0)
 extended palette --> 256-color tiles (with 16bit bgmap entries)(text,rot/scal)
```

Allocated VRAM is split into 4 slots of 8K each (32K used in total), normally BG0..3 are using Slot 0..3, however BG0 and BG1 can be optionally changed to BG0=Slot2, and BG1=Slot3 via BG0CNT and BG1CNT.

OBJ Extended Palette enabled in DISPCNT Bit 31, when enabled,

```
 16 colors x 16 palettes --> standard palette memory (=256 colors)
 256 colors x 16 palettes --> extended palette memory (=4096 colors)
```

Extended OBJ palette memory must be allocated to VRAM F, G, or I (which are 16K) of which only the first 8K are used for extended palettes (=1000h 16bit entries).

DS Video Capture and Main Memory Display Mode

4000064h - NDS9 - DISPCAPCNT - 32bit - Display Capture Control Register (R/W)

Capture is supported for Display Engine A only.

```
  0-4   EVA               (0..16 = Blending Factor for Source A)
  5-7   Not used
  8-12  EVB               (0..16 = Blending Factor for Source B)
  13-15 Not used
  16-17 VRAM Write Block  (0..3 = VRAM A..D) (VRAM must be allocated to LCDC)
  18-19 VRAM Write Offset (0=00000h, 0=08000h, 0=10000h, 0=18000h)
  20-21 Capture Size      (0=128x128, 1=256x64, 2=256x128, 3=256x192 dots)
  22-23 Not used
  24    Source A          (0=Graphics Screen BG+3D+OBJ, 1=3D Screen)
  25    Source B          (0=VRAM, 1=Main Memory Display FIFO)
  26-27 VRAM Read Offset  (0=00000h, 0=08000h, 0=10000h, 0=18000h)
  28    Not used
  29-30 Capture Source    (0=Source A, 1=Source B, 2/3=Sources A+B blended)
  31    Capture Enable    (0=Disable/Ready, 1=Enable/Busy)
```

Notes:
VRAM Read Block (VRAM A..D) is selected in DISPCNT Bits 18-19.
VRAM Read Block can be (or must be ?) allocated to LCDC (MST=0).
VRAM Read Offset is ignored (zero) in VRAM Display Mode (DISPCNT.16-17).
VRAM Read/Write Offsets wrap to 00000h when exceeding 1FFFFh (max 128K).
Capture Sizes less than 256x192 capture the upper-left portion of the screen.
Blending factors EVA and EVB are used only if "Source A+B blended" selected.
After setting the Capture Enable bit, capture starts at next line 0, and the capture enable/busy bit is then automatically cleared (in line 192, regardless of the capture size).

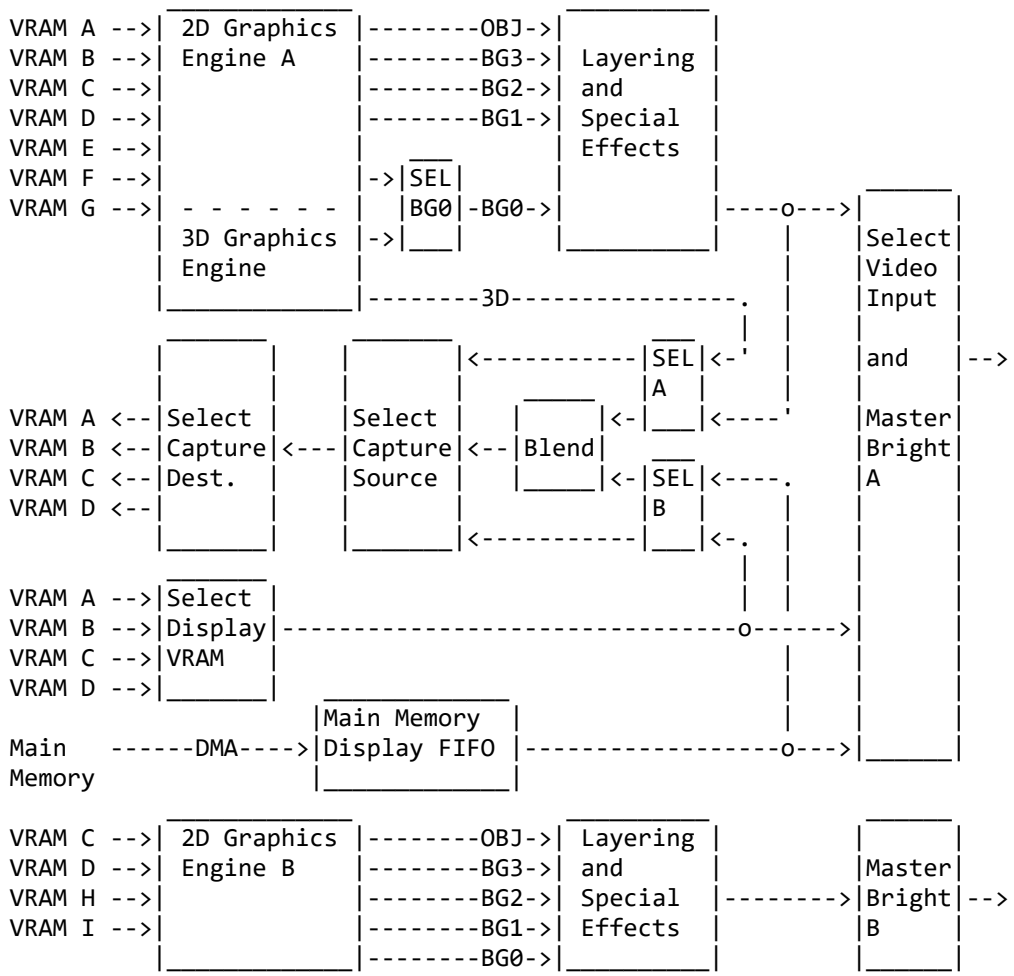
Capture data is 15bit color depth (even when capturing 18bit 3D-images).
Capture A: Dest_Intensity = SrcA_Intensitity ; Dest_Alpha=SrcA_Alpha.
Capture B: Dest_Intensity = SrcB_Intensitity ; Dest_Alpha=SrcB_Alpha.
Capture A+B (blending):

```
 Dest_Intensity = (  (SrcA_Intensitity * SrcA_Alpha * EVA)
                   + (SrcB_Intensitity * SrcB_Alpha * EVB) ) / 16
 Dest_Alpha = (SrcA_Alpha AND (EVA>0)) OR (SrcB_Alpha AND EVB>0))
```

Capture provides a couple of interesting effects.
For example, 3D Engine output can be captured via source A (to LCDC-allocated VRAM), in the next frame, either Graphics Engine A or B can display the captured 3D image in VRAM image as BG2, BG3, or OBJ (from BG/OBJ-allocated VRAM); this method requires to switch between LCDC- and BG/OBJ-allocation.
Another example would be to capture Engine A output, the captured image can be displayed (via VRAM Display mode) in the following frames, simultaneously the new Engine A output can be captured, blended with the old captured image; in that mode moved objects will leave traces on the screen; this method works with a single LCDC-allocated VRAM block.
DS Video Display System Block Diagram

4000068h - NDS9 - DISP_MMEM_FIFO - 32bit - Main Memory Display FIFO (R?/W)
Intended to send 256x192 pixel 32K color bitmaps by DMA directly
```
 - to Screen A             (set DISPCNT to Main Memory Display mode), or
 - to Display Capture unit (set DISPCAPCNT to Main Memory Source).
```
The FIFO can receive 4 words (8 pixels) at a time, each pixel is a 15bit RGB value (the upper bit, bit15, is unused).
Set DMA to Main Memory mode, 32bit transfer width, word count set to 4, destination address to DISP_MMEM_FIFO, source address must be in Main Memory.
Transfer starts at next frame.
Main Memory Display/Capture is supported for Display Engine A only.

DS Video Display System Block Diagram



DS 3D Video

3D is more or less (about 92%) understood and described.

DS 3D Overview

The NDS 3D hardware consists of a Geometry Engine, and a Rendering Engine.

Geometry Engine (Precalculate coordinates & assign polygon attributes)
Geometry commands can be sent via Ports 4000440h and up (or alternately, written directly to Port 4000400h). The commands include matrix and vector multiplications, the purpose is to rotate/scale/translate coordinates (vertices), the resulting coordinates are stored in Vertex RAM.
Moreover, it allows to assign attributes to the polygons and vertices, that includes vertex colors (or automatically calculated light colors), texture attributes, number of vertices per polygon (three or four), and a number of flags, these attributes are stored in Polygon RAM. Polygon RAM also contains pointers to the corresponding vertices in Vertex RAM.

Swap Buffers (Pass data from the Geometry Engine to the Rendering Engine)
The hardware includes two sets of Vertex/Polygon RAM, one used by the Geometry Engine, one by the Rendering Engine. The SwapBuffers command simply exchanges these buffers (so the new Geometry Data is passed to the Rendering Engine) (and the old buffer is emptied, so the Geometry engine can write new data to it). Additionally, the two parameter bits from the <previous> SwapBuffers command are copied to the Geometry Engine.
Data that is NOT swapped: SwapBuffers obviously can't swap Texture memory (so software must take care that Texture memory is kept mapped throughout rendering). Moreover, the rendering control registers (ports 4000060h, and 4000330h..40003BFh) are not swapped (so that values must be kept intact during rendering, too).

Rendering Engine (Display Output)

The Rendering Engine draws the various Polygons, and outputs them as BG0 layer to the 2D Video controller (which may then output them to the screen, or to the video capture unit). The Rendering part is done automatically by hardware, so the software has little influence on it.
Rendering is done scanline-by-scanline, so there's only a limited number of clock cycles per scanline, which is limiting the maximum number of polygons per scanline. However, due to the 48-line cache (see below), some scanlines are allowed to exceed that maximum.
Rendering starts 48 lines in advance (while still in the Vblank period) (and does then continue throughout the whole display period), the rendered data is written to a small cache that can hold up to 48 scanlines.

Scanline Cache vs Framebuffer
Note: There's only the 48-line cache (not a full 192-line framebuffer to store the whole rendered image). That is perfectly reasonable since animated data is normally drawn only once (so there would be no need to store it). That, assuming that the Geometry Engine presents new data every frame (otherwise, if the Geometry software is too slow, or if the image isn't animated, then the hardware is automatically rendering the same image again, and again).

DS 3D I/O Map

3D I/O Map

Address	Siz	Name	Expl.
Rendering Engine (per Frame settings)			
4000060h	2	DISP3DCNT	3D Display Control Register (R/W)
4000320h	1	RDLINES_COUNT	Rendered Line Count Register (R)
4000330h	10h	EDGE_COLOR	Edge Colors 0..7 (W)
4000340h	1	ALPHA_TEST_REF	Alpha-Test Comparision Value (W)
4000350h	4	CLEAR_COLOR	Clear Color Attribute Register (W)
4000354h	2	CLEAR_DEPTH	Clear Depth Register (W)
4000356h	2	CLRIMAGE_OFFSET	Rear-plane Bitmap Scroll Offsets (W)
4000358h	4	FOG_COLOR	Fog Color (W)
400035Ch	2	FOG_OFFSET	Fog Depth Offset (W)
4000360h	20h	FOG_TABLE	Fog Density Table, 32 entries (W)
4000380h	40h	TOON_TABLE	Toon Table, 32 colors (W)
Geometry Engine (per Polygon/Vertex settings)			
4000400h	40h	GXFIFO	Geometry Command FIFO (W)
4000440h	...	...	Geometry Command Ports (see below)
4000600h	4	GXSTAT	Geometry Engine Status Register (R and R/W)
4000604h	4	RAM_COUNT	Polygon List & Vertex RAM Count Register (R)
4000610h	2	DISP_1DOT_DEPTH	1-Dot Polygon Display Boundary Depth (W)
4000620h	10h	POS_RESULT	Position Test Results (R)
4000630h	6	VEC_RESULT	Vector Test Results (R)
4000640h	40h	CLIPMTX_RESULT	Read Current Clip Coordinates Matrix (R)
4000680h	24h	VECMTX_RESULT	Read Current Directional Vector Matrix (R)

Geometry Commands (can be invoked by Port Address, or by Command ID)
Table shows Port Address, Command ID, Number of Parameters, and Clock Cycles.

Address	Cmd	Pa.	Cy.	
N/A	00h	-	-	NOP - No Operation (for padding packed GXFIFO commands)
4000440h	10h	1	1	MTX_MODE - Set Matrix Mode (W)
4000444h	11h	-	17	MTX_PUSH - Push Current Matrix on Stack (W)
4000448h	12h	1	36	MTX_POP - Pop Current Matrix from Stack (W)
400044Ch	13h	1	17	MTX_STORE - Store Current Matrix on Stack (W)
4000450h	14h	1	36	MTX_RESTORE - Restore Current Matrix from Stack (W)
4000454h	15h	-	19	MTX_IDENTITY - Load Unit Matrix to Current Matrix (W)
4000458h	16h	16	34	MTX_LOAD_4x4 - Load 4x4 Matrix to Current Matrix (W)
400045Ch	17h	12	30	MTX_LOAD_4x3 - Load 4x3 Matrix to Current Matrix (W)
4000460h	18h	16	35*	MTX_MULT_4x4 - Multiply Current Matrix by 4x4 Matrix (W)
4000464h	19h	12	31*	MTX_MULT_4x3 - Multiply Current Matrix by 4x3 Matrix (W)
4000468h	1Ah	9	28*	MTX_MULT_3x3 - Multiply Current Matrix by 3x3 Matrix (W)
400046Ch	1Bh	3	22	MTX_SCALE - Multiply Current Matrix by Scale Matrix (W)

```
4000470h 1Ch 3  22* MTX_TRANS - Mult. Curr. Matrix by Translation Matrix (W)
4000480h 20h 1  1   COLOR - Directly Set Vertex Color (W)
4000484h 21h 1  9*  NORMAL - Set Normal Vector (W)
4000488h 22h 1  1   TEXCOORD - Set Texture Coordinates (W)
400048Ch 23h 2  9   VTX_16 - Set Vertex XYZ Coordinates (W)
4000490h 24h 1  8   VTX_10 - Set Vertex XYZ Coordinates (W)
4000494h 25h 1  8   VTX_XY - Set Vertex XY Coordinates (W)
4000498h 26h 1  8   VTX_XZ - Set Vertex XZ Coordinates (W)
400049Ch 27h 1  8   VTX_YZ - Set Vertex YZ Coordinates (W)
40004A0h 28h 1  8   VTX_DIFF - Set Relative Vertex Coordinates (W)
40004A4h 29h 1  1   POLYGON_ATTR - Set Polygon Attributes (W)
40004A8h 2Ah 1  1   TEXIMAGE_PARAM - Set Texture Parameters (W)
40004ACh 2Bh 1  1   PLTT_BASE - Set Texture Palette Base Address (W)
40004C0h 30h 1  4   DIF_AMB - MaterialColor0 - Diffuse/Ambient Reflect. (W)
40004C4h 31h 1  4   SPE_EMI - MaterialColor1 - Specular Ref. & Emission (W)
40004C8h 32h 1  6   LIGHT_VECTOR - Set Light's Directional Vector (W)
40004CCh 33h 1  1   LIGHT_COLOR - Set Light Color (W)
40004D0h 34h 32 32  SHININESS - Specular Reflection Shininess Table (W)
4000500h 40h 1  1   BEGIN_VTXS - Start of Vertex List (W)
4000504h 41h -  1   END_VTXS - End of Vertex List (W)
4000540h 50h 1  392 SWAP_BUFFERS - Swap Rendering Engine Buffer (W)
4000580h 60h 1  1   VIEWPORT - Set Viewport (W)
40005C0h 70h 3  103 BOX_TEST - Test if Cuboid Sits inside View Volume (W)
40005C4h 71h 2  9   POS_TEST - Set Position Coordinates for Test (W)
40005C8h 72h 1  5   VEC_TEST - Set Directional Vector for Test (W)
```

All cycle timings are counted in 33.51MHz units. NORMAL commands takes 9..12 cycles, depending on the number of enabled lights in PolyAttr (Huh, 9..12 (four timings) cycles for 0..4 (five settings) lights?) Total execution time of SwapBuffers is Duration until VBlank, plus 392 cycles.
In MTX_MODE=2 (Simultanous Set), MTX_MULT/TRANS take additional 30 cycles.

DS 3D Display Control

4000060h - DISP3DCNT - 3D Display Control Register (R/W)

```
0     Texture Mapping      (0=Disable, 1=Enable)
1     PolygonAttr Shading  (0=Toon Shading, 1=Highlight Shading)
2     Alpha-Test           (0=Disable, 1=Enable) (see ALPHA_TEST_REF)
3     Alpha-Blending       (0=Disable, 1=Enable) (see various Alpha values)
4     Anti-Aliasing        (0=Disable, 1=Enable)
5     Edge-Marking         (0=Disable, 1=Enable) (see EDGE_COLOR)
6     Fog Color/Alpha Mode (0=Alpha and Color, 1=Only Alpha) (see FOG_COLOR)
7     Fog Master Enable    (0=Disable, 1=Enable)
8-11  Fog Depth Shift      (FOG_STEP=400h shr FOG_SHIFT) (see FOG_OFFSET)
12    Color Buffer RDLINES Underflow (0=None, 1=Underflow/Acknowledge)
13    Polygon/Vertex RAM Overflow    (0=None, 1=Overflow/Acknowledge)
14    Rear-Plane Mode                (0=Blank, 1=Bitmap)
15-31 Not used
```

4000540h - Cmd 50h - SWAP_BUFFERS - Swap Rendering Engine Buffer (W)

SwapBuffers exchanges the two sets of Polygon/Vertex RAM buffers, that is, the newly defined polygons/vertices are passed to the rendering engine (and will be displayed in following frame(s)). The other buffer is emptied, and passed to the Geometry Engine (to be filled with new polygons/vertices by Geometry Commands).

```
0     Translucent polygon Y-sorting (0=Auto-sort, 1=Manual-sort)
1     Depth Buffering  (0=With Z-value, 1=With W-value)
      (mode 1 does not function properly with orthogonal projections)
2-31  Not used
```

SwapBuffers isn't executed until next VBlank (Scanline 192) (the Geometry Engine is halted for that duration).
SwapBuffers should not be issued within Begin/End. The two parameter bits of the SwapBuffers command are used for the following gxcommands (ie. not for the old gxcommands prior to SwapBuffers).

SwapBuffers does lock-up the 3D hardware if an incomplete polygon list has been defined (eg. a triangle with only 2 vertices). On lock-up, only 2D video is kept working, any wait-loops for GXSTAT.27 will hang the program. Once lock-up has occured, there seems to be no way to recover by software, not by sending the missing veric(es), and not even by pulsing POWCNT1.Bit2-3.

4000580h - Cmd 60h - VIEWPORT - Set Viewport (W)

```
  0-7   Screen/BG0 Coordinate X1 (0..255) (For Fullscreen: 0=Left-most)
  8-15  Screen/BG0 Coordinate Y1 (0..191) (For Fullscreen: 0=Bottom-most)
  16-23 Screen/BG0 Coordinate X2 (0..255) (For Fullscreen: 255=Right-most)
  24-31 Screen/BG0 Coordinate Y2 (0..191) (For Fullscreen: 191=Top-most)
```

Coordinate 0,0 is the lower-left (unlike for 2D where it'd be upper-left).
The 3D view-volume (size as defined by the Projection Matrix) is automatically scaled to match into the Viewport area. Although polygon vertices are clipped to the view-volume, some vertices may still exceed to X2,Y1 (lower-right) boundary by one pixel, due to some sort of rounding errors. The Viewport settings don't affect the size or position of the 3D Rear-Plane. Viewport should not be issued within Begin/End.

4000610h - DISP_1DOT_DEPTH - 1-Dot Polygon Display Boundary Depth (W)

1-Dot Polygons are very small, or very distant polygons, which would be rendered as a single pixel on screen. Polygons with a depth value greater (more distant) than DISP_1DOT_DEPTH can be automatically hidden; in order to reduce memory consumption, or to reduce dirt on the screen.

```
  0-14  W-Coordinate (Unsigned, 12bit integer, 3bit fractional part)
  15-31 Not used                 (0000h=Closest, 7FFFh=Most Distant)
```

The DISP_1DOT_DEPTH comparision can be enabled/disabled per polygon (via POLYGON_ATTR.Bit13), so "important" polygons can be displayed regardless of their size and distance.
Note: The comparision is always using the W-coordinate of the vertex (not the Z-coordinate) (ie. no matter if using Z-buffering, or W-buffering). The polygon is rendered if at least one of its vertices is having a w-coordinate less or equal than DISP_1DOT_DEPTH. NB. despite of checking the w-coords of ALL vertices, the polygon is rendered using the color/depth/texture of its FIRST vertex.
Note: The hardware does round-up the width and height of all polygons to at least 1, so polygons of 0x0, 1x0, 0x1, and 1x1 dots will be all rounded-up to a size of 1x1. Of which, the so-called "1dot" depth check is applied only to the 0x0 dot variant (so "0dot" depth check would be a better name for it).
Caution: Although DISP_1DOT_DEPTH is a Geometry Engine parameter, it is NOT routed through GXFIFO, ie. changes will take place immediately, and will affect all following polygons, including such that are still in GXFIFO. Workaround: ensure that GXFIFO is empty before changing this parameter.

4000340h - ALPHA_TEST_REF - Alpha-Test Comparision Value (W)

Alpha Test can be enabled in DISP3DCNT.Bit2. When enabled, pixels are rendered only if their Alpha value is GREATER than ALPHA_TEST_REF. Otherwise, when disabled, pixels are rendered only if their Alpha value is GREATER than zero. Alpha Test is performed on the final polygon pixels (ie. after texture blending).

```
  0-4   Alpha-Test Comparision Value (0..31) (Draw pixels if Alpha>AlphaRef)
  5-31  Not used
```

Value 00h is effectively the same as when Alpha Test is disabled. Value 1Fh hides all polygons, including opaque ones.

DS 3D Geometry Commands

4000400h - GXFIFO - Geometry Command FIFO (W) (mirrored up to 400043Fh?)

Used to send packed commands, unpacked commands,

```
  0-7   First  Packed Command (or Unpacked Command)
  8-15  Second Packed Command (or 00h=None)
  16-23 Third  Packed Command (or 00h=None)
  24-31 Fourth Packed Command (or 00h=None)
```

and parameters,

```
  0-31  Parameter data for the previously sent (packed) command(s)
```

to the Geometry engine.

FIFO / PIPE Number of Entries
The FIFO has 256 entries, additionally, there is a PIPE with four entries (giving a total of 260 entries). If the FIFO is empty, and if the PIPE isn't full, then data is moved directly into the PIPE, otherwise it is moved into the FIFO. If the PIPE runs half empty (less than 3 entries) then 2 entries are moved from the FIFO to the PIPE. The state of the FIFO can be obtained in GXSTAT.Bit16-26, observe that there may be still data in the PIPE, even if the FIFO is empty. Check the busy flag in GXSTAT.Bit27 to see if the PIPE or FIFO contains data (or if a command is still executing).
Each PIPE/FIFO entry consists of 40bits of data (8bit command code, plus 32bit parameter value). Commands without parameters occupy 1 entry, and Commands with N parameters occupy N entries.

Sending Commands by Ports 4000440h..40005FFh
Geometry commands can be indirectly sent to the FIFO via ports 4000440h and up.
For a command with N paramters: issue N writes to the port.
For a command without parameters: issue one dummy-write to the port.
That mechanism puts the 8bit command + 32bit parameter into the FIFO/PIPE.
If the FIFO is full, then a wait is generated until data is removed from the FIFO, ie. the STR opcode gets freezed, during the wait, the bus cannot be used even by DMA, interrupts, or by the NDS7 CPU.

GXFIFO Access via DMA
Larger pre-calculated data blocks can be sent directly to the FIFO. This is usually done via DMA (use DMA in Geometry Command Mode, 32bit units, Dest=4000400h/fixed, Length=NumWords, Repeat=0). The timings are handled automatically, ie. the system (should) doesn't freeze when the FIFO is full (see below Overkill note though). DMA starts when the FIFO becomes less than half full, the DMA does then write 112 words to the GXFIFO register (or less, if the remaining DMA transfer length gets zero).

GXFIFO Access via STR,STRD,STM
If desired, STR,STRD,STM opcodes can be used to write to the FIFO.
Opcodes that write more than one 32bit value (ie. STRD and STM) can be used to send ONE UNPACKED command, plus any parameters which belong to that command. After that, there must be a 1 cycle delay before sending the next command (ie. one cannot sent more than one command at once with a single opcode, each command must be invoked by a new opcode). STRD and STM can be used because the GXFIFO register is mirrored to 4000400h..43Fh (16 words).
As with Ports 4000440h and up, the CPU gets stopped if (and as long as) the FIFO is full.

GXFIFO / Unpacked Commands
```
- command1 (upper 24bit zero)
- parameter(s) for command1 (if any)
- command2 (upper 24bit zero)
- parameter(s) for command2 (if any)
- command3 (upper 24bit zero)
- parameter(s) for command3 (if any)
```

GXFIFO / Packed Commands
```
- command1,2,3,4 packed into one 32bit value (all bits used)
- parameter(s) for command1 (if any)
- parameter(s) for command2 (if any)
- parameter(s) for command3 (if any)
- parameter(s) for command4 (top-most packed command MUST have parameters)
- command5,6 packed into one 32bit value (upper 16bit zero)
- parameter(s) for command5 (if any)
- parameter(s) for command6 (top-most packed command MUST have parameters)
- command7,8,9 packed into one 32bit value (upper 8bit zero)
- parameter(s) for command7 (if any)
- parameter(s) for command8 (if any)
- parameter(s) for command9 (top-most packed command MUST have parameters)
```

Packed commands are first decompressed and then stored in command the FIFO.

GXFIFO DMA Overkill on Packed Commands Without Parameters
Normally, the 112 word limit ensures that the FIFO (256 entries) doesn't get full, however, this limit is much too high for sending a lot of "Packed Commands Without Parameters" (ie. PUSH, IDENTITY, or END) - eg. sending 112 x Packed(00151515h) to GXFIFO would write 336 x Cmd(15h) to the FIFO, which is causing the FIFO to get full, and which is causing the DMA (and CPU) to be paused (for several seconds, in WORST case) until enough FIFO commands have been processed to allow the DMA to finish the 112 word transfer.
Not sure if there's much chance to get Overkills in practice. Normally most commands DO have parameters, and so, usually even LESS than 112 FIFO entries are occupied (since 8bit commands with 32bit parameters are merged into single 40bit FIFO entries).

Invalid GX commands
Invalid commands (anything else than 10h..1Ch, 20h..2Bh, 30h..33h, 40h..41h, 50h, 60h, or 70h..72h) seem to be simply ignored by the hardware (at least, testing has confirmed that they do not fetch any parameters from the gxfifo).

DS 3D Matrix Load/Multiply

4000440h - Cmd 10h - MTX_MODE - Set Matrix Mode (W)
```
  0-1   Matrix Mode (0..3)
         0  Projection Matrix
         1  Position Matrix (aka Modelview Matrix)
         2  Position & Vector Simultaneous Set mode (used for Light+VEC_TEST)
         3  Texture Matrix (see DS 3D Texture Coordinates chapter)
  2-31  Not used
```
Selects the current Matrix, all following MTX commands (load, multiply, push, pop, etc.) are applied to that matrix. In Mode 2, all MTX commands are applied to both the Position and Vector matrices. There are two special cases:
```
  MTX_SCALE in Mode 2:                  uses ONLY Position Matrix
  MTX_PUSH/POP/STORE/RESTORE in Mode 1: uses BOTH Position AND Vector Matrices
```
Ie. the four stack commands act like mode 2 (even when in mode 1; keeping the two stacks somewhat in sync), and scale acts like mode 1 (even when in mode 2; keeping the light vector length's intact).

```
 vice-versa for the scale command.
```
For the above cases, the commands do always act like mode 1, even when they are i

4000454h - Cmd 15h - MTX_IDENTITY - Load Unit Matrix to Current Matrix (W)
Sets C=I. Parameters: None
The Identity Matrix (I), aka Unit Matrix, consists of all zeroes, with a diagonal row of ones. A matrix multiplied by the Unit Matrix is left unchanged.

4000458h - Cmd 16h - MTX_LOAD_4x4 - Load 4x4 Matrix to Current Matrix (W)
Sets C=M. Parameters: 16, m[0..15]

400045Ch - Cmd 17h - MTX_LOAD_4x3 - Load 4x3 Matrix to Current Matrix (W)
Sets C=M. Parameters: 12, m[0..11]

4000460h - Cmd 18h - MTX_MULT_4x4 - Multiply Current Matrix by 4x4 Matrix (W)
Sets C=M*C. Parameters: 16, m[0..15]

4000464h - Cmd 19h - MTX_MULT_4x3 - Multiply Current Matrix by 4x3 Matrix (W)
Sets C=M*C. Parameters: 12, m[0..11]

4000468h - Cmd 1Ah - MTX_MULT_3x3 - Multiply Current Matrix by 3x3 Matrix (W)
Sets C=M*C. Parameters: 9, m[0..8]

400046Ch - Cmd 1Bh - MTX_SCALE - Multiply Current Matrix by Scale Matrix (W)
Sets C=M*C. Parameters: 3, m[0..2]
Note: MTX_SCALE doesn't change Vector Matrix (even when in MTX_MODE=2) (that's done so for keeping the length of the light vector's intact).

4000470h - Cmd 1Ch - MTX_TRANS - Mult. Curr. Matrix by Translation Matrix (W)
Sets C=M*C. Parameters: 3, m[0..2] (x,y,z position)

4000640h..67Fh - CLIPMTX_RESULT - Read Current Clip Coordinates Matrix (R)
This 64-byte region (16 words) contains the m[0..15] values of the Current Clip Coordinates Matrix, arranged in 4x4 Matrix format. Make sure that the Geometry Engine is stopped (GXSTAT.27) before reading from these registers.
The Clip Matrix is internally used to convert vertices to screen coordinates, and is internally re-calculated anytime when changing the Position or Projection matrices:

```
ClipMatrix = PositionMatrix * ProjectionMatrix
```

To read only the Position Matrix, or only the Projection Matrix: Use Load Identity on the OTHER matrix, so the ClipMatrix becomes equal to the DESIRED matrix (multiplied by the Identity Matrix, which has no effect on the result).

4000680h..6A3h - VECMTX_RESULT - Read Current Directional Vector Matrix (R)
This 36-byte region (9 words) contains the m[0..8] values of the Current Directional Vector Matrix, arranged in 3x3 Matrix format (the fourth row/column may contain any values).
Make sure that the Geometry Engine is stopped (GXSTAT.27) before reading from these registers.

DS 3D Matrix Types

Essentially, all matrices in the NDS are 4x4 Matrices, consisting of 16 values, m[0..15]. Each element is a signed fixed-point 32bit number, with a fractional part in the lower 12bits.
The other Matrix Types are used to reduce the number of parameters being transferred, for example, 3x3 Matrix requires only nine parameters, the other seven elements are automatically set to 0 or 1.0 (whereas "1.0" means "1 SHL 12" in 12bit fixed-point notation).

```
 _      4x4 Matrix       _        _     Identity Matrix    _
| m[0]  m[1]  m[2]  m[3]  |      |  1.0   0     0     0    |
| m[4]  m[5]  m[6]  m[7]  |      |  0     1.0   0     0    |
| m[8]  m[9]  m[10] m[11] |      |  0     0     1.0   0    |
|_m[12] m[13] m[14] m[15]_|      |_ 0     0     0     1.0 _|

 _      4x3 Matrix       _        _  Translation Matrix   _
| m[0]  m[1]  m[2]   0    |      |  1.0   0     0     0    |
| m[3]  m[4]  m[5]   0    |      |  0     1.0   0     0    |
| m[6]  m[7]  m[8]   0    |      |  0     0     1.0   0    |
|_m[9]  m[10] m[11]  1.0 _|      |_m[0]  m[1]  m[2]   1.0 _|

 _      3x3 Matrix       _        _      Scale Matrix      _
| m[0]  m[1]  m[2]   0    |      | m[0]   0     0     0    |
| m[3]  m[4]  m[5]   0    |      |  0    m[1]   0     0    |
| m[6]  m[7]  m[8]   0    |      |  0     0    m[2]   0    |
|_ 0     0     0     1.0 _|      |_ 0     0     0     1.0 _|
```

DS 3D Matrix Stack

Matrix Stack
The NDS has three Matrix Stacks, and two Matrix Stack Pointers (the Coordinate Matrix stack pointer is also shared for Directional Matrix Stack).

```
Matrix Stack________Valid Stack Area____Stack Pointer____________________
Projection Stack    0..0  (1 entry)     0..1  (1bit) (GXSTAT: 1bit)
Coordinate Stack    0..30 (31 entries)  0..63 (6bit) (GXSTAT: 5bit only)
Directional Stack   0..30 (31 entries)  (uses Coordinate Stack Pointer)
Texture Stack       One..None?          0..1  (1bit) (GXSTAT: N/A)
```

Which of the stacks/matrices depends on the current Matrix Mode (as usually, but with one exception; stack operations MTX_PUSH/POP/STORE/RESTORE in Mode 1 are acting same as in Mode 2):

```
MTX_MODE = 0      --> Projection Stack
MTX_MODE = 1 or 2 --> BOTH Coordinate AND Directional Stack
MTX_MODE = 3      --> Texture Stack
```

The initial value of the Stack Pointers is zero, the current value of the pointers can be read from GXSTAT (read-only), that register does also indicate stack overflows (errors flag gets set on read/write to invalid entries, ie. entries 1 or 1Fh..3Fh). For all stacks, the upper half (ie. 1 or 20h..3Fh) are mirrors of the lower half (ie. 0 or 0..1Fh).

4000444h - Cmd 11h - MTX_PUSH - Push Current Matrix on Stack (W)
Parameters: None. Sets [S]=C, and then S=S+1.

4000448h - Cmd 12h - MTX_POP - Pop Current Matrix from Stack (W)
Sets S=S-N, and then C=[S].

```
Parameter Bit0-5:  Stack Offset (signed value, -30..+31) (usually +1)
Parameter Bit6-31: Not used
```

Offset N=(+1) pops the most recently pushed value, larger offsets of N>1 will "deallocate" N values (and load the Nth value into C). Zero or negative values can be used to pop previously "deallocated" values.
The stack has only one level (at address 0) in projection mode, in that mode, the parameter value is ignored, the offset is always +1 in that mode.

400044Ch - Cmd 13h - MTX_STORE - Store Current Matrix on Stack (W)
Sets [N]=C. The stack pointer S is not used, and is left unchanged.

```
Parameter Bit0-4:  Stack Address (0..30) (31 causes overflow in GXSTAT.15)
Parameter Bit5-31: Not used
```

The stack has only one level (at address 0) in projection mode, in that mode, the parameter value is ignored.

4000450h - Cmd 14h - MTX_RESTORE - Restore Current Matrix from Stack (W)
Sets C=[N]. The stack pointer S is not used, and is left unchanged.

```
Parameter Bit0-4:  Stack Address (0..30) (31 causes overflow in GXSTAT.15)
Parameter Bit5-31: Not used
```

The stack has only one level (at address 0) in projection mode, in that mode, the parameter value is ignored.

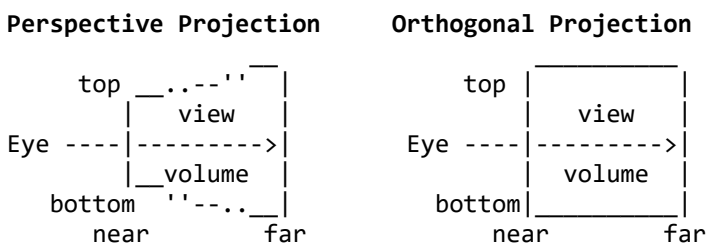
In Projection mode, the parameter for POP, STORE, and RESTORE is unused - not sure if the parameter (ie. a dummy value) is - or is not - to be written to the command FIFO?
There appear to be actually 32 entries in Coordinate & Directional Stacks, entry 31 appears to exist, and appears to be read/write-able (although the stack overflow flag gets set when accessing it).

DS 3D Matrix Examples (Projection)

The most important matrix is the Projection Matrix (to be initialized with MTX_MODE=0 via MTX_LOAD_4x4 command). It does specify the dimensions of the view volume.

With Perspective Projections more distant objects will appear smaller, with Orthogonal Projects the size of the

objects is always same regardless of their distance.



Correctly initializing the projection matrix (as shown in the examples below) can be quite difficult (mind that fixed point multiply/divide requires to adjust the fixed-point width before/after calculation). For beginners, it may be recommended to start with a simple Identity Matrix (MTX_IDENTITY command) used as Projection Matrix (ie. Ortho with t,b,l,r set to +/-1).

Orthogonal Projections (Ortho)

```
| (2.0)/(r-l)       0             0             0     |
|      0       (2.0)/(t-b)        0             0     |
|      0            0        (2.0)/(n-f)       0     |
| (l+r)/(l-r)  (b+t)/(b-t)   (n+f)/(n-f)      1.0    |
```

n,f specify the distance from eye to near and far clip planes. t,b,l,r are the coordinates of near clip plane (top,bottom,left,right). For a symmetrical view (ie. the straight-ahead view line centered in the middle of viewport) t,b,l,r should be usually t=+ysiz/2, b=-ysiz/2, r=+xsiz/2, l=-xsiz/2; the (xsiz/ysiz) ratio should be usually equal to the viewport's (width/heigh) ratio. Examples for a asymmetrical view would be b=0 (frog's view), or t=0 (bird's view).

Left-Right Asymmetrical Perspective Projections (Frustum)

```
| (2*n)/(r-l)       0             0             0     |
|      0       (2*n)/(t-b)        0             0     |
| (r+l)/(r-l)  (t+b)/(t-b)   (n+f)/(n-f)     -1.0    |
|      0            0       (2*n*f)/(n-f)      0     |
```

n,f,t,b,l,r have same meanings as above (Ortho), the difference is that more distant objects will appear smaller with Perspective Projection (unlike Orthogonal Projection where the size isn't affected by the distance).

Left-Right Symmetrical Perspective Projections (Perspective)

```
| cos/(asp*sin)     0             0             0     |
|      0         cos/sin          0             0     |
|      0            0        (n+f)/(n-f)     -1.0    |
|      0            0       (2*n*f)/(n-f)      0     |
```

Quite the same as above (Frustum), but with symmetrical t,b values (which are in this case obtained from a vertical view range specified in degrees), and l,r are matched to the aspect ratio of the viewport (asp=height/width).

Moving the Camera

After initializing the Projection Matrix, you may multiply it with Rotate and/or Translation Matrices to change camera's position and view direction.

DS 3D Matrix Examples (Rotate/Scale/Translate)

Identity Matrix

The MTX_IDENTITY command can be used to initialize the Position Matrix before doing any Translation/Scaling/Rotation, for example:

```
Load(Identity)                          ;no rotation/scaling used
Load(Identity), Mul(Rotate), Mul(Scale)  ;rotation/scaling (not so efficient)
Load(Rotate), Mul(Scale)                ;rotation/scaling (more efficient)
```

Rotation Matrices
Rotation can be performed with MTX_MULT_3x3 command, simple examples are:

```
  Around X-Axis          Around Y-Axis          Around Z-Axis
  | 1.0  0     0   |     | cos   0    sin |     | cos   sin   0   |
  | 0    cos   sin |     | 0     1.0  0   |     | -sin  cos   0   |
  | 0    -sin  cos |     | -sin  0    cos |     | 0     0     1.0 |
```

Scale Matrix
The MTX_SCALE command allows to adjust the size of the polygon. The x,y,z parameters should be normally all having the same value, x=y=z (unless if you want to change only the height of the object, for example). Identical results can be obtained with MTX_MULT commands, however, when using lighting (MTX_MODE=2), then scaling should be done ONLY with MTX_SCALE (which keeps the length of the light's directional vector intact).

Translation Matrix
The MTX_TRANS command allows to move polygons to the desired position. The polygon VTX commands are spanning only a small range of coordinates (near zero-coordinate), so translation is required to move the polygons to other locations in the world coordinates. Aside from that, translation is useful for moved objects (at variable coordinates), and for re-using an object at various locations (eg. you can create a forest by translating a tree to different coordinates).

Matrix Multiply Order
The Matrix must be set up BEFORE sending the Vertices (which are then automatically multiplied by the matrix). When using multiple matrices multiplied with each other: Mind that, for matrix maths A*B is NOT the same as B*A. For example, if you combine Rotate and Translate Matrices, the object will be either rotated around it's own zero-coordinate, or around world-space zero-coordinate, depending on the multiply order.

DS 3D Matrix Examples (Maths Basics)

Below is a crash-course on matrix maths. Most of it is carried out automatically by the hardware. So this chapter is relevant only if you are interested in details about what happens inside of the 3D engine.

Matrix-by-Matrix Multiplication
Matrix multiplication, C = A * B, is possible only if the number of columns in A is equal to the number of rows in B, so it works fine with the 4x4 matrices which are used in the NDS. For the multiplication, assume matrix C to consist of elements cyx, and respecitively, matrix A and B to consist of elements ayx and byx. So that C = A * B looks like:

```
  | c11 c12 c13 c14 |     | a11 a12 a13 a14 |     | b11 b12 b13 b14 |
  | c21 c22 c23 c24 |  =  | a21 a22 a23 a24 |  *  | b21 b22 b23 b24 |
  | c31 c32 c33 c34 |     | a31 a32 a33 a34 |     | b31 b32 b33 b34 |
  | c41 c42 c43 c44 |     | a41 a42 a43 a44 |     | b41 b42 b43 b44 |
```

Each element in C is calculated by multiplying the elements from one row in A by the elements from the corresponding column in B, and then taking the sum of the products, ie.

```
  cyx = ay1*b1x + ay2*b2x + ay3*b3x + ay4*b4x
```

In total, that requires 64 multiplications (four multiplications for each of the 16 cyx elements), and 48 additions (three per cyx element), the hardware carries out that operation at a relative decent speed of 30..35 clock cycles, possibly by performing several multiplications simultaneously with separate multiply units.
Observe that for matrix multiplication, A*B is NOT the same as B*A.

Matrix-by-Vector & Vector-by-Matrix Multiplication
Vectors are Matrices with only one row, or only one column. Multiplication works as for normal matrices; the number of rows/columns must match up, repectively, row-vectors can be multiplied by matrices; and matrices can be multiplied by column-vectors (but not vice-versa). Eg. C = A * B:

```
                                                  | b11 b12 b13 b14 |
| c11 c12 c13 c14 |  =  | a11 a12 a13 a14 |  *  | b21 b22 b23 b24 |
                                                  | b31 b32 b33 b34 |
                                                  | b41 b42 b43 b44 |
```

The formula for calculating the separate elements is same as above,

```
cyx = ay1*b1x + ay2*b2x + ay3*b3x + ay4*b4x
```

Of which, C and A have only one y-index, so one may replace "cyx and ayx" by "c1x and a1x", or completely leave out the y-index, ie. "cx and ax".

Matrix-by-Number Multiplication

Simply multiply all elements of the Matrix by the number, C = A * n:

```
cyx = ayx*n
```

Of course, works also with vectors (matrices with only one row/column).

Matrix-to-Matrix Addition/Subtraction

Both matrices must have the same number of rows & columns, add/subtract all elements with corresponding elements in other matrix, C = A +/- B:

```
cyx = ayx +/- byx
```

Of course, works also with vectors (two matrices with only one row/column).

Vectors

A vector, for example (x,y,z), consists of offsets along x-,y-, and z-axis. The line from origin to origin-plus-offset is having two characteristics: A direction, and a length.
The length (aka magnitude) can be calculated as L=sqrt(x^2+y^2+z^2).

Vector-by-Vector Multiplication

This can be processed as LineVector*RowVector, so the result is a number (aka scalar) (aka a matrix with only 1x1 elements). Multiplying two (normalized) vectors results in: "cos(angle)=vec1*vec2", ie. the cosine of the angle between the two vectors (eg. used for light vectors). Multiplying a vector with itself, and taking the square root of the result obtains its length, ie. "length=sqrt(vec^2)".
That stuff should be done with 3-dimensional vectors (not 4-dimensionals).

Normalized Vectors

Normalized Vectors (aka Unit Vectors) are vectors with length=1.0. To normalize a vector, divide its coordinates by its length, ie. x=x/L, y=y/L, z=z/L, the direction remains the same, but the length is now 1.0.
On the NDS, normalized vectors should have a length of something less than 1.0 (eg. something like 0.99) because several NDS registers are limited to 1bit sign, 0bit integer, Nbit fractional part (so vectors that are parallel to the x,y,z axes, or that become parallel to them after rotation, cannot have a length of 1.0).

Fixed-Point Numbers

The NDS uses fixed-point numbers (rather than floating point numbers). Addition and Subtraction works as with normal integers, provided that the fractional part is the same for both numbers. If it is not the same: Shift-left the value with the smaller fractional part.
For multiplication, the fractional part of result is the sum of the fractional parts (eg. 12bit fraction * 12bit fraction = 24bit fraction; shift-right the result by 12 to convert it 12bit fraction). The NDS matrix multiply unit is maintaining the full 24bit fraction when processing the

```
cyx = ay1*b1x + ay2*b2x + ay3*b3x + ay4*b4x
```

formula, ie. the three additions are using full 24bit fractions (with carry-outs to upper bits), the final result of the additions is then shifted-right by 12.
For division, it's vice versa, the fractions of the operands are substracted, 24bit fraction / 12bit fraction = 12bit fraction. When dividing two 12bit numbers, shift-left the first number by 12 before division to get a result with 12bit fractional part.

Four-Dimensional Matrices

The NDS uses four-dimensional matrices and vectors, ie. matrices with 4x4 elements, and vectors with 4

elements. The first three elements are associated with the X,Y,Z-axes of the three-dimensional space. The fourth element is somewhat a "W-axis".
With 4-dimensional matrices, the Translate matrix can be used to move an object to another position. Ie. once when you've setup a matrix (which may consists of pre-multiplied scaling, rotation, translation matrices), then that matrix can be used on vertices to perform the rotation, scaling, translation all-at-once; by a single Vector*Matrix operation.
With 3-dimensional matrices, translation would require a separate addition, additionally to the multiply operation.

DS 3D Polygon Attributes

40004A4h - Cmd 29h - POLYGON_ATTR - Set Polygon Attributes (W)

```
  0-3   Light 0..3 Enable Flags (each bit: 0=Disable, 1=Enable)
  4-5   Polygon Mode  (0=Modulation,1=Decal,2=Toon/Highlight Shading,3=Shadow)
  6     Polygon Back Surface   (0=Hide, 1=Render)  ;Line-segments are always
  7     Polygon Front Surface  (0=Hide, 1=Render)  ;rendered (no front/back)
  8-10  Not used
  11    Depth-value for Translucent Pixels    (0=Keep Old, 1=Set New Depth)
  12    Far-plane intersecting polygons       (0=Hide, 1=Render/clipped)
  13    1-Dot polygons behind DISP_1DOT_DEPTH (0=Hide, 1=Render)
  14    Depth Test, Draw Pixels with Depth    (0=Less, 1=Equal) (usually 0)
  15    Fog Enable                            (0=Disable, 1=Enable)
  16-20 Alpha      (0=Wire-Frame, 1..30=Translucent, 31=Solid)
  21-23 Not used
  24-29 Polygon ID (00h..3Fh, used for translucent, shadow, and edge-marking)
  30-31 Not used
```

Writes to POLYGON_ATTR have no effect until next BEGIN_VTXS command.
Changes to the Light bits have no effect until lighting is re-calculated by Normal command. The interior of Wire-frame polygons is transparent (Alpha=0), and only the lines at the polygon edges are rendered, using a fixed Alpha value of 31.

4000480h - Cmd 20h - COLOR - Directly Set Vertex Color (W)

```
  Parameter 1, Bit 0-4    Red
  Parameter 1, Bit 5-9    Green
  Parameter 1, Bit 10-14  Blue
  Parameter 1, Bit 15-31  Not used
```

The 5bit RGB values are internally expanded to 6bit RGB as follows: X=X*2+(X+31)/32, ie. zero remains zero, all other values are X=X*2+1.
Aside from by using the Color command, the color can be also changed by MaterialColor0 command (if MaterialColor0.Bit15 is set, it acts identical as the Color Command), and by the Normal command (which calculates the color based on light/material parameters).

Depth Test

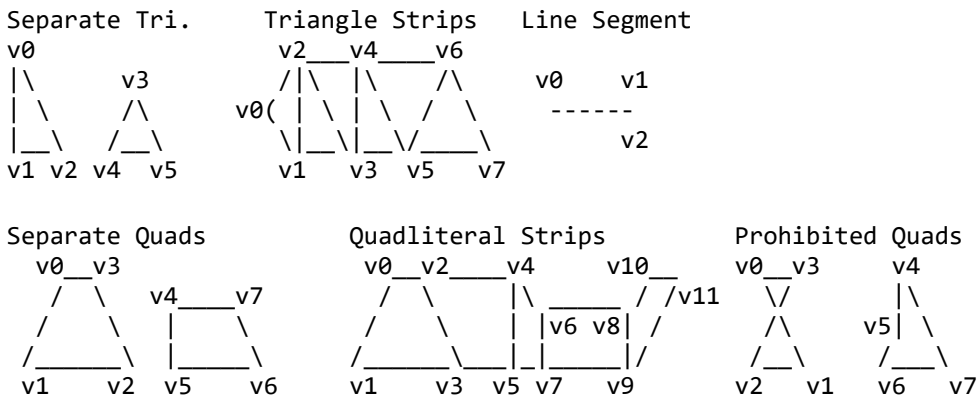
The Depth Test compares the depth of the pixels of the polygon with the depth of previously rendered polygons (or of the rear plane if there have been none rendered yet). The new pixels are drawn if the new depth is Less (closer to the camera), or if it is Equal, as selected by POLYGON_ATTR.Bit14.
Normally, Depth Equal would work only exact matches (ie. if the overlapping polygons have exactly the same coordinates; and thus have the same rounding errors), however, the NDS hardware is allowing "Equal" to have a tolerance of +/-200h (within the 24bit depth range of 0..FFFFFFh), that may bypass rounding errors, but it may also cause nearby polygons to be accidently treated to have equal depth.

DS 3D Polygon Definitions by Vertices

The DS supports polygons with 3 or 4 edges, triangles and quadliterals.

The position of the edges is defined by vertices, each consisting of (x,y,z) values.

For Line Segments, use Triangles with twice the same vertex, Line Segments are rendered always because they do not have any front and back sides.
The Prohibited Quad shapes may produce unintended results, namely, that are Quads with crossed sides, and quads with angles greater than 180 degrees.



The vertices are normally arranged anti-clockwise, except that: in triangle-strips each second polygon uses clockwise arranged vertices, and quad-strips are sorts of "up-down" arranged (whereas "up" and "down" may be anywhere due to rotation). Other arrangements may result in quads with crossed lines, or may swap the front and back sides of the polygon (above examples are showing the front sides).

4000500h - Cmd 40h - BEGIN_VTXS - Start of Vertex List (W)
```
  Parameter 1, Bit 0-1    Primitive Type (0..3, see below)
  Parameter 1, Bit 2-31   Not used
```
Indicates the Start of a Vertex List, and its Primitive Type:
```
  0  Separate Triangle(s)    ;3*N vertices per N triangles
  1  Separate Quadliteral(s) ;4*N vertices per N quads
  2  Triangle Strips         ;3+(N-1) vertices per N triangles
  3  Quadliteral Strips      ;4+(N-1)*2 vertices per N quads
```
The BEGIN_VTX command should be followed by VTX_-commands to define the Vertices of the list, and should be then terminated by END_VTX command.
BEGIN_VTX additionally applies changes to POLYGON_ATTR.

4000504h - Cmd 41h - END_VTXS - End of Vertex List (W)
Parameters: None. This is a Dummy command for OpenGL compatibility. It should be used to terminate a BEGIN_VTX, VTX_<values> sequence. END_VTXS is possibly required for Nintendo's software emulator? On real NDS consoles (and in no$gba) it does have no effect, it can be left out, or can be issued multiple times inside of a vertex list, without disturbing the display.

400048Ch - Cmd 23h - VTX_16 - Set Vertex XYZ Coordinates (W)
```
  Parameter 1, Bit 0-15   X-Coordinate (signed, with 12bit fractional part)
  Parameter 1, Bit 16-31  Y-Coordinate (signed, with 12bit fractional part)
  Parameter 2, Bit 0-15   Z-Coordinate (signed, with 12bit fractional part)
  Parameter 2, Bit 16-31  Not used
```

4000490h - Cmd 24h - VTX_10 - Set Vertex XYZ Coordinates (W)
```
  Parameter 1, Bit 0-9    X-Coordinate (signed, with 6bit fractional part)
  Parameter 1, Bit 10-19  Y-Coordinate (signed, with 6bit fractional part)
  Parameter 1, Bit 20-29  Z-Coordinate (signed, with 6bit fractional part)
  Parameter 1, Bit 30-31  Not used
```
Same as VTX_16, with only one parameter, with smaller fractional part.

4000494h - Cmd 25h - VTX_XY - Set Vertex XY Coordinates (W)

```
  Parameter 1, Bit 0-15   X-Coordinate (signed, with 12bit fractional part)
  Parameter 1, Bit 16-31  Y-Coordinate (signed, with 12bit fractional part)
```
The Z-Coordinate is kept unchanged, and re-uses the value from previous VTX.

4000498h - Cmd 26h - VTX_XZ - Set Vertex XZ Coordinates (W)
```
  Parameter 1, Bit 0-15   X-Coordinate (signed, with 12bit fractional part)
  Parameter 1, Bit 16-31  Z-Coordinate (signed, with 12bit fractional part)
```
The Y-Coordinate is kept unchanged, and re-uses the value from previous VTX.

400049Ch - Cmd 27h - VTX_YZ - Set Vertex YZ Coordinates (W)
```
  Parameter 1, Bit 0-15   Y-Coordinate (signed, with 12bit fractional part)
  Parameter 1, Bit 16-31  Z-Coordinate (signed, with 12bit fractional part)
```
The X-Coordinate is kept unchanged, and re-uses the value from previous VTX.

40004A0h - Cmd 28h - VTX_DIFF - Set Relative Vertex Coordinates (W)
```
  Parameter 1, Bit 0-9    X-Difference (signed, with 9/12bit fractional part)
  Parameter 1, Bit 10-19  Y-Difference (signed, with 9/12bit fractional part)
  Parameter 1, Bit 20-29  Z-Difference (signed, with 9/12bit fractional part)
  Parameter 1, Bit 30-31  Not used
```
Sets XYZ-Coordinate relative to the XYZ-Coordinates from previous VTX. In detail: The 9bit fractional values are divided by 8 (sign expanded to 12bit fractions, in range +/-0.125), and that 12bit fraction is then added to the old vtx coordinates. The result of the addition should not overflow 16bit vertex coordinate range (1bit sign, 3bit integer, 12bit fraction).

Notes on VTX commands
On each VTX command, the viewport coordinates of the vertex are calculated and stored in Vertex RAM,
```
  ( xx, yy, zz, ww ) = ( x, y, z, 1.0 ) * ClipMatrix
```
The actual screen position (in pixels) is then,
```
  screen_x = (xx+ww)*viewport_width / (2*ww) + viewport_x1
  screen_y = (yy+ww)*viewport_height / (2*ww) + viewport_y1
```
Each VTX command that completes the definition of a polygon (ie. each 3rd for Separate Trangles) does additionally store data in Polygon List RAM.
VTX commands may be issued only between Begin and End commands.

Clipping
Polygons are clipped to the 6 sides of the view volume (ie. to the left, right, top, bottom, near, and far edges). If one or more vertic(es) exceed one of these sides, then these vertic(es) are replaced by two newly created vertices (which are located on the intersections of the polygon edges and the view volume edge).
Depending on the number of clipped vertic(es), this may increase or decrease the number of entries in Vertex RAM (ie. minus N clipped vertices, plus 2 new vertices). Also, clipped polygons which are part of polygon strips are converted to separate polygons (which does increase number of entries in Vertex RAM). Polygons that are fully outside of the View Volume aren't stored in Vertex RAM, nor in Polygon RAM (the only exception are polygons that are located exactly one pixel below of, or right of lower/right edges, which appear to be accidently stored in memory).

DS 3D Polygon Light Parameters

The lighting operation is performed by executing the Normal command (which sets the VertexColor based on the Light/Material parameters) (to the rest of the hardware it doesn't matter if the VertexColor was set by Color command or by Normal command). Light is calculated only for the Front side of the polygon (assuming that the Normal is matched to that side), so the Back side will be (incorrectly) using the same color.

40004C8h - Cmd 32h - LIGHT_VECTOR - Set Light's Directional Vector (W)
Sets direction of the specified light (ie. the light selected in Bit30-31).
```
  0-9   Directional Vector's X component (1bit sign + 9bit fractional part)
```

```
  10-19 Directional Vector's Y component (1bit sign + 9bit fractional part)
  20-29 Directional Vector's Z component (1bit sign + 9bit fractional part)
  30-31 Light Number                     (0..3)
```
Upon executing this command, the incoming vector is multiplied by the current Directional Matrix, the result is then applied as LightVector. This allows to rotate the light direction. However, normally, to keep the light unrotated, be sure to use LoadIdentity (in MtxMode=2) before setting the LightVector.

40004CCh - Cmd 33h - LIGHT_COLOR - Set Light Color (W)
Sets the color of the specified light (ie. the light selected in Bit30-31).
```
  0-4   Red          (0..1Fh)      ;\light color this will be combined with
  5-9   Green        (0..1Fh)      ; diffuse, specular, and ambient colors
  10-14 Blue         (0..1Fh)      ;/upon execution of the normal command
  15-29 Not used
  30-31 Light Number (0..3)
```

40004C0h - Cmd 30h - DIF_AMB - MaterialColor0 - Diffuse/Ambient Reflect. (W)
```
  0-4   Diffuse Reflection Red     ;\light(s) that directly hits the polygon,
  5-9   Diffuse Reflection Green   ; ie. max when NormalVector has opposite
  10-14 Diffuse Reflection Blue    ;/direction of LightVector
  15    Set Vertex Color (0=No, 1=Set Diffuse Reflection Color as Vertex Color)
  16-20 Ambient Reflection Red     ;\light(s) that indirectly hits the polygon,
  21-25 Ambient Reflection Green   ; ie. assuming that light is reflected by
  26-30 Ambient Reflection Blue    ;/walls/floor, regardless of LightVector
  31    Not used
```
With Bit15 set, the lower 15bits are applied as VertexColor (exactly as when when executing the Color command), the purpose is to use it as default color (eg. when outcommenting the Normal command), normally, when using lighting, the color setting gets overwritten (as soon as executing the Normal command).

40004C4h - Cmd 31h - SPE_EMI - MaterialColor1 - Specular Ref. & Emission (W)
```
  0-4   Specular Reflection Red    ;\light(s) reflected towards the camera,
  5-9   Specular Reflection Green  ; ie. max when NormalVector is in middle of
  10-14 Specular Reflection Blue   ;/LightVector and ViewDirection
  15    Specular Reflection Shininess Table (0=Disable, 1=Enable)
  16-20 Emission Red               ;\light emitted by the polygon itself,
  21-25 Emission Green             ; ie. regardless of light colors/vectors,
  26-30 Emission Blue              ;/and no matter if any lights are enabled
  31    Not used
```
Caution: Specular Reflection WON'T WORK when the ProjectionMatrix is rotated.

40004D0h - Cmd 34h - SHININESS - Specular Reflection Shininess Table (W)
Write 32 parameter words (each 32bit word containing four 8bit entries), entries 0..3 in the first word, through entries 124..127 in the last word:
```
  0-7   Shininess 0 (unsigned fixed-point, 0bit integer, 8bit fractional part)
  8-15  Shininess 1 ("")
  16-23 Shininess 2 ("")
  24-31 Shininess 3 ("")
```
If the table is disabled (by MaterialColor1.Bit15), then reflection will act as if the table would be filled with linear increasing numbers.

4000484h - Cmd 21h - NORMAL - Set Normal Vector (W)
In short, this command does calculate the VertexColor, based on the various light-parameters.
In detail, upon executing this command, the incoming vector is multiplied by the current Directional Matrix, the result is then applied as NormalVector (giving it the same rotation as used for the following polygon vertices).
```
  0-9   X-Component of Normal Vector (1bit sign + 9bit fractional part)
  10-19 Y-Component of Normal Vector (1bit sign + 9bit fractional part)
  20-29 Z-Component of Normal Vector (1bit sign + 9bit fractional part)
  30-31 Not used
```
Defines the Polygon's Normal. And, does then update the Vertex Color; by recursing the View Direction, the NormalVector, the LightVector(s), and Light/Material Colors. The execution time of the Normal command

varies depending on the number of enabled light(s).

Additional Light Registers
Additionally to above registers, light(s) must be enabled in PolygonAttr (mind that changes to PolygonAttr aren't applied until next Begin command). And, the Directional Matrix must be set up correctly (in MtxMode=2) for the LightVector and NormalVector commands.

Normal Vector
The Normal vector must point "away from the polygon surface" (eg. for the floor, the Normal should point upwards). That direction is implied by the polygon vertices, however, the hardware cannot automatically calculate it, so it must be set manually with the Normal command (prior to the VTX-commands).
When using lighting, the Normal command must be re-executed after switching Lighting on/off, or after changing light/material parameters. And, of course, also before defining polygons with different orientation. Polygons with same orientation (eg. horizontal polygon surfaces) and same material color can use the same Normal. Changing the Normal per polygon gives differently colored polygons with flat surfaces, changing the Normal per vertex gives the illusion of curved surfaces.

Light Vector
Each light consists of parallel beams; similar to sunlight, which appears to us (due to the great distance) to consist of parallel beams, all emmitted into the same direction; towards Earth.
In reality, light is emitted into ALL directions, originated from the light source (eg. a candle), the hardware doesn't support that type of non-parallel light. However, the light vectors can be changed per polygon, so a polygon that is located north of the light source may use different light direction than a polygon that is east of the light source.
And, of course, Light 0..3 may (and should) have different directions.

Normalized Vectors
The Normal Vector and the Light Vectors should be normalized (ie. their length should be 1.0) (in practice: something like 0.99, since the registers have only fractional parts) (a length of 1.0 can cause overflows).

Lighting Limitations
The functionality of the light feature is limited to reflecting light to the camera (light is not reflected to other polygons, nor does it cast shadows on other polygons). However, independently of the lighting feature, the DS hardware does allow to create shadows, see:
DS 3D Shadow Polygons

Internal Operation on Normal Command
```
IF TexCoordTransformMode=2 THEN TexCoord=NormalVector*Matrix (see TexCoord)
NormalVector=NormalVector*DirectionalMatrix
VertexColor = EmissionColor
FOR i=0 to 3
 IF PolygonAttrLight[i]=enabled THEN
  DiffuseLevel = max(0,-(LightVector[i]*NormalVector))
  ShininessLevel = max(0,(-HalfVector[i])*(NormalVector))^2
  IF TableEnabled THEN ShininessLevel = ShininessTable[ShininessLevel]
  ;note: below processed separately for the R,G,B color components...
  VertexColor = VertexColor + SpecularColor*LightColor[i]*ShininessLevel
  VertexColor = VertexColor + DiffuseColor*LightColor[i]*DiffuseLevel
  VertexColor = VertexColor + AmbientColor*LightColor[i]
 ENDIF
NEXT i
```

Internal Operation on Light_Vector Command (for Light i)
```
LightVector[i] = (LightVector*DirectionalMatrix)
HalfVector[i] = (LightVector[i]+LineOfSightVector)/2
```

LineOfSightVector (how it SHOULD work)

Ideally, the LineOfSightVector should point from the camera to the vertic(es), however, the vertic(es) are still unknown at time of normal command, so it is just pointing from the camera to the screen, ie.

```
LineOfSightVector = (0,0,-1.0)
```

Moreover, the LineOfSightVector should be multiplied by the Projection Matrix (so the vector would get rotated accordingly when the camera gets rotated), and, after multiplication by a scaled matrix, it'd be required to normalize the resulting vector.

LineOfSightVector (how it DOES actually work)

However, the NDS cannot normalize vectors by hardware, and therefore, it does completely leave out the LineOfSightVector*ProjectionMatrix multiplication. So, the LineOfSightVector is always (0,0,-1.0), no matter of any camera rotation. That means,

```
Specular Reflection WON'T WORK when the ProjectionMatrix is rotated (!)
```

So, if you want to rotate the "camera" (in MTX_MODE=0), then you must instead rotate the "world" in the opposite direction (in MTX_MODE=2).

That problem applies only to Specular Reflection, ie. only if Lighting is used, and only if the Specular Material Color is nonzero.

Maths Notes

Note on Vector*Vector multiplication: Processed as LineVector*RowVector, so the result is a number (aka scalar) (aka a matrix with only 1x1 elements), multiplying two (normalized) vectors results in: "cos(angle)=vec1*vec2", ie. the cosine of the angle between the two vectors.

The various Normal/Light/Half/Sight vectors are only 3-dimensional (x,y,z), ie. only the upper-left 3x3 matrix elements are used on multiplications with the 4x4 DirectionalMatrix.

DS 3D Shadow Polygons

The DS hardware's Light-function allows to reflect light to the camera, it does not reflect light to other polygons, and it does not cast any shadows. For shadows at fixed locations it'd be best to pre-calculate their shape and position, and to change the vertex color of the shaded polygons.

Additionally, the Shadow Polygon feature can be used to create animated shadows, ie. moved objects and variable light sources.

Shadow Polygons and Shadow Volume

The software must define a Shadow Volume (ie. the region which doesn't contain light), the hardware does then automatically draw the shadow on all pixels whose x/y/z-coordinates are inside of that region.

The Shadow Volume must be defined by several Shadow Polygons which are enclosing the shaded region. The 'top' of the shadow volume should be usually translated to the position of the object that casts the shadow, if the light direction changes then the shadow volume should be also rotated to match the light direction. The 'length' of the shadow volume should be (at least) long enough to reach from the object to the walls/floor where the shadow is to be drawn. The shadow volume must be passed TWICE to the hardware:

Step 1 - Shadow Volume for Mask

Set Polygon_Attr Mode=Shadow, PolygonID=00h, Back=Render, Front=Hide, Alpha=01h..1Eh, and pass the shadow volume (ie. the shadow polygons) to the geometry engine.

The Back=Render / Front=Hide setting causes the 'rear-side' of the shadow volume to be rendered, of course only as far as it is in front of other polygons. The Mode=Shadow / ID=00h setting causes the polygon NOT to be drawn to the Color Buffer - instead, flags are set in the Stencil Buffer (to be used in Step 2).

Step 2 - Shadow Volume for Rendering

Simply repeat step 1, but with Polygon_Attr Mode=Shadow, PolygonID=01h..3Fh, Back=Render(what/why?), Front=Render, Alpha=01h..1Eh.

The Front=Render setting causes the 'front-side' of the shadow volume to be rendered, again, only as far as it is in front of other polygons. The Mode=Shadow / ID>00h setting causes the polygon to be drawn to the Color

Buffer as usually, but only if the Stencil Buffer bits are zero (ie. the portion from Step 1 is excluded) (additionally, Step 2 resets the stencil bits after checking them). Moreover, the shadow is rendered only if its Polygon ID differs from the ID in the Attribute Buffer.

Shadow Alpha and Shadow Color
The Alpha=Translucent setting in Step 1 and 2 ensures that the Shadow is drawn AFTER the normal (opaque) polygons have been rendered. In Step 2 it does additionally specify the 'intensity' of the shadow. For normal shadows, the Vertex Color should be usually black, however, the shadow volume may be also used as 'spotlight volume' when using other colors.

Rendering Order
The Mask Volume must be rendered prior to the Rendering Volume, ie. Step 1 and 2 must be performed in that order, and, to keep that order intact, Auto-sorting must have been disabled in the previous Swap_Buffers command.
The shadow volume must be rendered after the 'target' polygons have been rendered, for opaque targets this is done automatically (due to the translucent alpha setting; translucent polygons are always rendered last, even with auto-sort disabled).

Translucent Targets
Casting shadows on Translucent Polygons. First draw the translucent target (with update depth buffer enabled, required for the shadow z-coordinates), then draw the Shadow Mask/Rendering volumes.
Due to the updated depth buffer the shadow will be cast only on the translucent target (not on any other polygons underneath of the translucent polygon). If you want the shadow to appear on both: Draw draw the Shadow Mask/Rendering volume TWICE (once before, and once after drawing the translucent target).

Polygon ID and Fog Enable
The "Render only if Polygon ID differs" feature (see Step 2) allows to prevent the shadow to be cast on the object that casts the shadow (ie. the object and shadow should have the same IDs). The feature also allows to select whether overlapping shadows (with same/different IDs) are shaded once or twice.
The old Fog Enable flag in the Attribute Buffer is ANDed with the Fog Enable flag of the Shadow Polygons, this allows to exclude Fog in shaded regions.

Shadow Volume Open/Closed Shapes
Normally, the shadow volume should have a closed shape, ie. should have rear-sides (step 1), and corresponding front-sides (step 2) for all possible viewing angles. That is required for the shadow to be drawn correctly, and also for the Stencil Buffer to be reset to zero (in step 2, so that the stencil bits won't disturb other shadow volumes).
Due to that, drawing errors may occur if the shadow volume's front or rear side gets clipped by near/far clip plane.
One exception is that the volume doesn't need a bottom-side (with a suitable volume length, the bottom may be left open, since it vanishes in the floor/walls anyways).

DS 3D Texture Attributes

4000488h - Cmd 22h - TEXCOORD - Set Texture Coordinates (W)
Specifies the texture source coordinates within the texture bitmap which are to be associated with the next vertex.

```
  Parameter 1, Bit 0-15   S-Coordinate (X-Coordinate in Texture Source)
  Parameter 1, Bit 16-31  T-Coordinate (Y-Coordinate in Texture Source)
  Both values are 1bit sign + 11bit integer + 4bit fractional part.
  A value of 1.0 (=1 SHL 4) equals to one Texel.
```

With Position 0.0 , 0.0 drawing starts from upperleft of the Texture.
With positive offsets, drawing origin starts more "within" the texture.

With negative offsets, drawing starts "before" the texture.
"When texture mapping, the Geometry Engine works faster if you issue commands in the order TexCoord -> Normal -> Vertex."

40004A8h - Cmd 2Ah - TEXIMAGE_PARAM - Set Texture Parameters (W)

```
0-15  Texture VRAM Offset div 8 (0..FFFFh -> 512K RAM in Slot 0,1,2,3)
      (VRAM must be allocated as Texture data, see Memory Control chapter)
16    Repeat in S Direction (0=Clamp Texture, 1=Repeat Texture)
17    Repeat in T Direction (0=Clamp Texture, 1=Repeat Texture)
18    Flip in S Direction   (0=No, 1=Flip each 2nd Texture) (requires Repeat)
19    Flip in T Direction   (0=No, 1=Flip each 2nd Texture) (requires Repeat)
20-22 Texture S-Size        (for N=0..7: Size=(8 SHL N); ie. 8..1024 texels)
23-25 Texture T-Size        (for N=0..7: Size=(8 SHL N); ie. 8..1024 texels)
26-28 Texture Format        (0..7, see below)
29    Color 0 of 4/16/256-Color Palettes (0=Displayed, 1=Made Transparent)
30-31 Texture Coordinates Transformation Mode (0..3, see below)
```

Texture Formats:

```
0  No Texture
1  A3I5 Translucent Texture
2  4-Color Palette Texture
3  16-Color Palette Texture
4  256-Color Palette Texture
5  4x4-Texel Compressed Texture
6  A5I3 Translucent Texture
7  Direct Texture
```

Texture Coordinates Transformation Modes:

```
0  Do not Transform texture coordinates
1  TexCoord source
2  Normal source
3  Vertex source
```

The S-Direction equals to the horizontal direction of the source bitmap.
The T-Direction, T-repeat, and T-flip are the same in vertical direction.
For a "/" shaped texture, the S-clamp, S-repeat, and S-flip look like so:

```
Clamp _____  Repeat       Repeat+Flip
_____/       ///////////  /\/\/\/\/\/
```

With "Clamp", the texture coordinates are clipped to MinMax(0,Size-1), so the texels at the edges of the texture bitmap are repeated (to avoid that effect, fill the bitmap edges by texels with alpha=0, so they become invisible).

40004ACh - Cmd 2Bh - PLTT_BASE - Set Texture Palette Base Address (W)

```
0-12   Palette Base Address (div8 or div10h, see below)
       (Not used for Texture Format 7: Direct Color Texture)
       (0..FFF8h/8 for Texture Format 2: ie. 4-color-palette Texture)
       (0..17FF0h/10h for all other Texture formats)
13-31  Not used
```

The palette data occupies 16bit per color, Bit0-4: Red, Bit5-9: Green, Bit10-14: Blue, Bit15: Not used.
(VRAM must be allocated as Texture Palette, there can be up to 6 Slots allocated, ie. the addressable 18000h bytes, see Memory Control chapter)

TexImageParam and TexPlttBase
Can be issued per polygon (except within polygon strips).

DS 3D Texture Formats

Format 2: 4-Color Palette Texture
Each Texel occupies 2bit, the first Texel is located in LSBs of 1st byte.
In this format, the Palette Base is specified in 8-byte steps; all other formats use 16-byte steps (see PLTT_BASE register).

Format 3: 16-Color Palette Texture
Each Texel occupies 4bit, the 1st Texel is located in LSBs of 1st byte.

Format 4: 256-Color Palette Texture
Each Texel occupies 8bit, the 1st Texel is located in 1st byte.

Format 7: Direct Color Texture
Each Texel occupies 16bit, the 1st Texel is located in 1st halfword.
Bit0-4: Red, Bit5-9: Green, Bit10-14: Blue, Bit15: Alpha

Format 1: A3I5 Translucent Texture (3bit Alpha, 5bit Color Index)
Each Texel occupies 8bit, the 1st Texel is located in 1st byte.

```
Bit0-4: Color Index (0..31) of a 32-color Palette
Bit5-7: Alpha       (0..7; 0=Transparent, 7=Solid)
```

The 3bit Alpha value (0..7) is internally expanded into a 5bit Alpha value (0..31) as follows: Alpha=(Alpha*4)+(Alpha/2).

Format 6: A5I3 Translucent Texture (5bit Alpha, 3bit Color Index)
Each Texel occupies 8bit, the 1st Texel is located in 1st byte.

```
Bit0-2: Color Index (0..7) of a 8-color Palette
Bit3-7: Alpha       (0..31; 0=Transparent, 31=Solid)
```

Format 5: 4x4-Texel Compressed Texture
Consists of 4x4 Texel blocks in Slot 0 or 2, 32bit per block, 2bit per Texel,

```
Bit0-7   Upper 4-Texel row (LSB=first/left-most Texel)
Bit8-15  Next  4-Texel row ("")
Bit16-23 Next  4-Texel row ("")
Bit24-31 Lower 4-Texel row ("")
```

Additional Palette Index Data for each 4x4 Texel Block is located in Slot 1,

```
Bit0-13  Palette Offset in 4-byte steps; Addr=(PLTT_BASE*10h)+(Offset*4)
Bit14-15 Transparent/Interpolation Mode (0..3, see below)
```

whereas, the Slot 1 offset is related to above Slot 0 or 2 offset,

```
slot1_addr = slot0_addr / 2           ;lower 64K of Slot1 assoc to Slot0
slot1_addr = slot2_addr / 2 + 10000h  ;upper 64K of Slot1 assoc to Slot2
```

The 2bit Texel values (0..3) are intepreted depending on the Mode (0..3),

Texel	Mode 0	Mode 1	Mode 2	Mode 3
0	Color 0	Color0	Color 0	Color 0
1	Color 1	Color1	Color 1	Color 1
2	Color 2	(Color0+Color1)/2	Color 2	(Color0*5+Color1*3)/8
3	Transparent	Transparent	Color 3	(Color0*3+Color1*5)/8

Mode 1 and 3 are using only 2 Palette Colors (which requires only half as much Palette memory), the 3rd (and 4th) Texel Colors are automatically set to above values (eg. to gray-shades if color 0 and 1 are black and white).
Note: The maximum size for 4x4-Texel Compressed Textures is 1024x512 or 512x1024 (which are both occupying the whole 128K in slot 0 or 2, plus 64K in slot1), a larger size of 1024x1024 cannot be used because of the gap between slot 0 and 2.

DS 3D Texture Coordinates

For textured polygons, a texture coordinate must be associated with each vertex of the polygon. The coordinates (S,T) are defined by TEXCOORD command (typically issued prior to each VTX command), and can be optionally automatically transformed, by the Transformation Mode selected in TEXIMAGE_PARAM register.

Texture Matrix
Although the texture matrix is 4x4, with values m[0..15], only the left two columns of this matrix are actually

used. In Mode 2 and 3, the bottom row of the matrix is replaced by S and T values from most recent TEXCOORD command.

Texture Coordinates Transformation Mode 0 - No Transform
The values are set upon executing the TEXCOORD command,

```
( S' T' ) = ( S  T )
```

Simple coordinate association, without using the Texture Matrix at all.

Texture Coordinates Transformation Mode 1 - TexCoord source
The values are calculated upon executing the TEXCOORD command,

```
                                  | m[0]  m[1]  |
( S' T' ) = ( S  T 1/16 1/16 ) * | m[4]  m[5]  |
                                  | m[8]  m[9]  |
                                  | m[12] m[13] |
```

Can be used to produce a simple texture scrolling, rotation, or scaling, by setting a translate, rotate, or scale matrix for the texture matrix.

Texture Coordinates Transformation Mode 2 - Normal source
The values are calculated upon executing the NORMAL command,

```
                                  | m[0]  m[1]  |
( S' T' ) = ( Nx  Ny  Nz 1.0 ) * | m[4]  m[5]  |
                                  | m[8]  m[9]  |
                                  | S     T     |
```

Can be used to produce spherical reflection mapping by setting the texture matrix to the current directional vector matrix, multiplied by a scaling matrix that expands the directional vector space from -1.0..+1.0 to one half of the texture size. For that purpose, translate the origin of the texture coordinate to the center of the spherical texture by using TexCoord command (spherical texture means a bitmap that contains some circle-shaped image).

Texture Coordinates Transformation Mode 3 - Vertex source
The values are calculated upon executing any VTX commands,

```
                                  | m[0]  m[1]  |
( S' T' ) = ( Vx  Vy  Vz 1.0 ) * | m[4]  m[5]  |
                                  | m[8]  m[9]  |
                                  | S     T     |
```

Can be used to produce texture scrolls dependent on the View coordinates by copying the current position coordinate matrix into the texture matrix. For example, the PositionMatrix can be obtained via CLIPMTX_RESULT (see there for details), and that values can be then manually copied to the TextureMatrix.

Sign+Integer+Fractional Parts used in above Formulas

```
Matrix    m[..]     1+19+12 (32bit)
Vertex    Vx,Vy,Vz  1+3+12  (16bit)
Normal    Nx,Ny,Nz  1+0+9   (10bit)
Constant  1.0       0+1+0   (1bit)
Constant  1/16      0+0+4   (4bit)
TexCoord  S,T       1+11+4  (16bit)
Result    S',T'     1+11+4  (16bit) <-------- clipped to that size !
```

Observe that the S',T' values are clipped to 16bit size. Ie. after the Vector*Matrix calaction, the result is shifted right (to make it having a 4bit fraction), and the value is then masked to 16bit size.

DS 3D Texture Blending

Polygon pixels consist of a Vertex Color, and of Texture Colors.
These colors can be blended as described below. Or, to use only either one:
To use only the Vertex Color: Select No Texture in TEXIMAGE_PARAM.
To use only the Texture Color: Select Modulation Mode and Alpha=31 in POLYGON_ATTR, and set COLOR to 7FFFh (white), or to gray values (to decrease brightness of the texture color).

Vertex Color (Rv,Gv,Bv,Av)
The Vertex Color (Rv,Gv,Bv) can be changed per Vertex (either by Color, Normal, or Material0 command), pixels between vertices are shaded to medium values of the surrounding vertices. The Vertex Alpha (Av), can be changed only per polygon (by PolygonAttr command).

Texture Colors (Rt,Gt,Bt,At)
The Texture Colors (Rt,Gt,Bt), and Alpha value (At), are defined by the Texture Bitmap. For formats without Alpha value, assume At=31 (solid), and for formats with 1bit Alpha assume At=A*31.

Shading Table Colors (Rs,Gs,Bs)
In Toon/Highlight Shading Mode, the red component of the Vertex Color (Rv) is mis-used as an index in the Shading Table, ie. Rv is used to read Shading Colors (Rs,Gs,Bs) from the table; the green and blue components of the Vertex Color (Gv,Bv) are unused in this mode. The Vertex Alpha (Av) is kept used.
Shading is used in Polygon Mode 2, whether it is Toon or Highlight Shading is selected in DISP3DCNT; this is a per-frame selection, so only either one can be used.

Texture Blending - Modulation Mode (Polygon Attr Mode 0)
```
R = ((Rt+1)*(Rv+1)-1)/64
G = ((Gt+1)*(Gv+1)-1)/64
B = ((Bt+1)*(Bv+1)-1)/64
A = ((At+1)*(Av+1)-1)/64
```
The multiplication result is decreased intensity (unless both factors are 63).

Texture Blending - Decal Mode (Polygon Attr Mode 1)
```
R = (Rt*At + Rv*(63-At))/64  ;except, when At=0: R=Rv, when At=31: R=Rt
G = (Gt*At + Gv*(63-At))/64  ;except, when At=0: G=Gv, when At=31: G=Gt
B = (Bt*At + Bv*(63-At))/64  ;except, when At=0: B=Bv, when At=31: B=Bt
A = Av
```
The At value is used (only) as ratio for Texture color vs Vertex Color.

Texture Blending - Toon Shading (Polygon Mode 2, DISP3DCNT=Toon)
The vertex color Red component (Rv) is used as an index in the toon table.
```
R = ((Rt+1)*(Rs+1)-1)/64   ;Rs=ToonTableRed[Rv]
G = ((Gt+1)*(Gs+1)-1)/64   ;Gs=ToonTableGreen[Rv]
B = ((Bt+1)*(Bs+1)-1)/64   ;Bs=ToonTableBlue[Rv]
A = ((At+1)*(Av+1)-1)/64
```
This is same as Modulation Mode, but using Rs,Gs,Bs instead Rv,Gv,Bv.

Texture Blending - Highlight Shading (Polygon Mode 2, DISP3DCNT=Highlight)
```
R = ((Rt+1)*(Rs+1)-1)/64+Rs ;truncated to MAX=63
G = ((Gt+1)*(Gs+1)-1)/64+Gs ;truncated to MAX=63
B = ((Bt+1)*(Bs+1)-1)/64+Bs ;truncated to MAX=63
A = ((At+1)*(Av+1)-1)/64
```
Same as Toon Shading, with additional addition offset, the addition may increase the intensity, however, it may also change the hue of the color.

Above formulas are for 6bit RGBA values, ie. 5bit values internally expanded to 6bit as such: IF X>0 THEN X=X*2+1.

Uni-Colored Textures
Although textures are normally containing "pictures", in some cases it makes sense to use "blank" textures that are filled with a single color:
Wire-frame polygons are always having Av=31, however, they can be made transparent by using Translucent Textures (ie. A5I3 or A3I5 formats) with At<31.
In Toon/Highlight shading modes, the Vertex Color is mis-used as table index, however, Toon/Highlight shading can be used on uni-colored textures, which is more or less the same as using Toon/Highlight shading on uni-

colored Vertex-colors.

DS 3D Toon, Edge, Fog, Alpha-Blending, Anti-Aliasing

4000380h..3BFh - TOON_TABLE - Toon Table (W)
This 64-byte region contains the 32 toon colors (16bit per color), used for both Toon and Highlight Shading. In both modes, the Red (R) component of the RGBA vertex color is mis-used as index to obtain the new RGB value from the toon table, vertex Alpha (A) is kept used as is.

```
Bit0-4: Red, Bit5-9: Green, Bit10-14: Blue, Bit15: Not Used
```

Shading can be enabled (per polygon) in Polygon_Attr, whether it is Toon or Highlight Shading is set (per frame) in DISP3DCNT. For more info on shading, see:
DS 3D Texture Blending

4000330h..33Fh - EDGE_COLOR - Edge Colors 0..7 (W)
This 16-byte region contains the 8 edge colors (16bit per color), Edge Color 0 is used for Polygon ID 00h..07h, Color 1 for ID 08h..0Fh, and so on.

```
Bit0-4: Red, Bit5-9: Green, Bit10-14: Blue, Bit15: Not Used
```

Edge Marking allows to mark the edges of an object (whose polygons all have the same ID) in a wire-frame style. Edge Marking can be enabled (per frame) in DISP3DCNT. When enabled, the polygon edges are drawn at the edge color, but only if the old ID value in the Attribute Buffer is different than the Polygon ID of the new polygon, so no edges are drawn between connected or overlapping polygons with same ID values.
Edge Marking is applied ONLY to opaque polygons (including wire-frames).
Edge Marking increases the size of opaque polygons (see notes below).
Edge Marking doesn't work very well with Anti-Aliasing (see Anti-Aliasing).
Technically, when rendering a polygon, it's edges (ie. the wire-frame region) are flagged as possible-edges (but it's still rendered normally, without using the edge-color). Once when all opaque polygons (*) have been rendered, the edge color is applied to these flagged pixels, under following conditions: At least one of the four surrounding pixels (up, down, left, right) must have different polygon_id than the edge, and, the edge depth must be LESS than the depth of that surrounding pixel (ie. no edges are rendered if the depth is GREATER or EQUAL, even if the polygon_id differs). At the screen borders, edges seem to be rendered in respect to the rear-plane's polygon_id entry (see Port 4000350h).
(*) Actually, edge-marking is reportedly performed not until all opaque AND translucent polygons have been rendered. That brings up some effects/problems when edges are covered by translucent polys: The edge-color is probably drawn as is (ie. it'll overwrite the translucent color, rather than being blended with the translucent color). And, any translucent polygons that do update the depth buffer will cause total edge-marking malfunction (since edge-marking involves the comparision of the current/surrounding pixel's depth values).

4000358h - FOG_COLOR - Fog Color (W)
Fog can be used to let more distant polygons to disappear in foggy grayness (or in darkness, or other color). This is particulary useful to "hide" the far clip plane. Fog can be enabled in DISP3DCNT.Bit7, moreover, when enabled, it can be activated or deactivated per polygon (POLYGON_ATTR.Bit15), and per Rear-plane (see there).

```
0-4    Fog Color, Red     ;\
5-9    Fog Color, Green   ; used only when DISP3DCNT.Bit6 is zero
10-14  Fog Color, Blue    ;/
15     Not used
16-20  Fog Alpha          ;-used no matter of DISP3DCNT.Bit6
21-31  Not used
```

Whether or not fog is applied to a pixel depends on the Fog flag in the framebuffer, the initial value of that flag can be defined in the rear-plane. When rendering opaque pixels, the framebuffer's fog flag gets replaced by PolygonAttr.Bit15. When rendering translucent pixels, the old flag in the framebuffer gets ANDed with PolygonAttr.Bit15.

400035Ch - FOG_OFFSET - Fog Depth Offset (W)

```
  0-14   Fog Offset (Unsigned) (0..7FFFh)
  15-31  Not used
```
FogDepthBoundary[0..31] (for FogDensity[0..31]) are defined as:
```
  FogDepthBoundary[n] = FOG_OFFSET + FOG_STEP*(n+1)   ;with n = 0..31
```
Whereas FOG_STEP is derived from the FOG_SHIFT value in DISP3DCNT.Bit8-11 (FOG_STEP=400h shr FOG_SHIFT) (normally FOG_SHIFT should be 0..10 (bigger shift amounts of 11..15 would cause FOG_STEP to become zero, so only Density[0] and Density[31] would be used).
The meaning of the depth values depends on whether z-values or w-values are stored in the framebuffer (see SwapBuffers.Bit1).
For translucent polygons, the depth value (and therefore: the amount of fog) depends on the depth update bit (see PolygonAttr.Bit11).

4000360h..37Fh - FOG_TABLE - Fog Density Table (W)
This 32-byte region contains FogDensity[0..31] (used at FogDepthBoundary[n]),
```
  0-6    Fog Density (00h..7Fh = None..Full) (usually increasing values)
  7      Not used
```
FogDensity[0] is used for all pixels closer than FogDepthBoundary[0], FogDensity[31] is used for all pixels more distant than FogDepthBoundary[0].
Density is linear interpolated for pixels that are between two Density depth boundaries. The formula for Fog Blending is:
```
  FrameBuffer[R] = (FogColor[R]*Density + FrameBuffer[R]*(128-Density)) / 128
  FrameBuffer[G] = (FogColor[G]*Density + FrameBuffer[G]*(128-Density)) / 128
  FrameBuffer[B] = (FogColor[B]*Density + FrameBuffer[B]*(128-Density)) / 128
  FrameBuffer[A] = (FogColor[A]*Density + FrameBuffer[A]*(128-Density)) / 128
```
If DISP3DCNT.Bit6 is set (=Alpha Only), then only FrameBuffer[A] is updated, and FrameBuffer[RGB] are kepth unchanged. Density=127 is handled as if Density=128.
Fog Glitch: The fog_alpha value appears to be ignored (treated as fog_alpha=1Fh) in the region up to the first density boundary. However, normally that value will be multiplied by zero (assumung that density[0] is usually zero), so you won't ever notice that hardware glitch.

Alpha-Blending (Polygon vs FrameBuffer)
Alpha-Blending occurs for pixels of translucent polygons,
```
  FrameBuf[R] = (Poly[R]*(Poly[A]+1) + FrameBuf[R]*(31-(Poly[A])) / 32
  FrameBuf[G] = (Poly[G]*(Poly[A]+1) + FrameBuf[G]*(31-(Poly[A])) / 32
  FrameBuf[B] = (Poly[B]*(Poly[A]+1) + FrameBuf[B]*(31-(Poly[A])) / 32
  FrameBuf[A] = max(Poly[A],FrameBuf[A])
```
There are three situations in which Alpha-Blending is bypassed (the old Framebuf[R,G,B,A] value is then simply overwritten by Poly[R,G,B,A]):
```
  1) Alpha-Blending is disabled                      (DISP3DCNT.Bit3=0)
  2) The polygon pixel is opaque                     (Poly[A]=31)
  3) The old framebuffer value is totally transparent (FrameBuf[A]=0)
```
The third case can happen if the rear-plane was initialized with Alpha=0, which causes the polygon not to be blended with the rear-plane (which may give better results when subsequently blending the 3D layer with the 2D engine).
Note: Totally transparent pixels (with Poly[A]=0) are not rendered (ie. neither FrameBuf[R,G,B,A] nor FrameBuf[Depth,Fog,PolyID,etc.] are updated.

Anti-Aliasing
Anti-Aliasing can be enabled in DISP3DCNT, when enabled, the edges of opaque polygons will be anti-aliased (ie. the pixels at the edges may become translucent).
Anti-Aliasing is not applied on translucent polygons. And, Anti-Aliasing is not applied on the interiors of the poylgons (eg. an 8x8 chessboard texture will be anti-aliased only at the board edges, not at the edges of the 64 fields).
Anti-Aliasing is (accidently) applied to opaque 1dot polygongs, line-segments and wire-frames (which results in dirty lines with missing pixels, 1dot polys become totally invisible), workaround is to use translucent dots, lines and wires (eg. with alpha=30).
Anti-Aliasing is (correctly) not applied to edges of Edge-Marked polygons, in that special case even opaque

line-segments and wire-frames are working even if anti-aliasing is enabled (provided that they are edge-marked, ie. if their polygon ID differs from the framebuffer's ID).
Anti-Aliasing is (accidently) making the edges of Edge-Marked polygons translucent (with alpha=16 or so?), that reduces the contrast of the edge colors. Moreover, if two of these translucent edges do overlap, then they are blended twice (even if they have the same polygon_id, and even if the depth_update bit in polygon_attr is set; both should normally prevent double-blending), that scatters the brightness of such edges.

Polygon Size
In some cases, the NDS hardware doesn't render the lower/right edges of certain polygons. That feature reduces rendering load, and, when rendering connected polygons (eg. strips), then it'd be unnecessary to render that edges (since they'd overlap with the upper/left edges of the other polygon). On the contrary, if there's no connected polygon displayed, then the polygon may appear smaller than expected. Small polygons with excluded edges are:

```
Opaque polygons (except wire-frames) without Edge-Marking and Anti-Aliasing,
and, all polygons with vertical right-edges (except line-segments).
Plus, Translucent Polys when Alpha-Blending is disabled in DISP3DCNT.Bit3.
```

All other polygons are rendered at full size with all edges included (except vertical right edges). Note: To disable the small-polygon feature, you can enable edge-marking (which does increase the polygon size, even if no edges are drawn, ie. even if all polys do have the same ID).

DS 3D Status

4000600h - GXSTAT - Geometry Engine Status Register (R and R/W)
Bit 30-31 are R/W. Writing "1" to Bit15 does reset the Error Flag (Bit15), and additionally resets the Projection Stack Pointer (Bit13), and probably (?) also the Texture Stack Pointer. All other GXSTAT bits are read-only.

```
 0     BoxTest,PositionTest,VectorTest Busy (0=Ready, 1=Busy)
 1     BoxTest Result  (0=All Outside View, 1=Parts or Fully Inside View)
 2-7   Not used
 8-12  Position & Vector Matrix Stack Level (0..31) (lower 5bit of 6bit value)
 13    Projection Matrix Stack Level        (0..1)
 14    Matrix Stack Busy (0=No, 1=Yes; Currently executing a Push/Pop command)
 15    Matrix Stack Overflow/Underflow Error (0=No, 1=Error/Acknowledge/Reset)
 16-24 Number of 40bit-entries in Command FIFO  (0..256)
(24)   Command FIFO Full (MSB of above)  (0=No, 1=Yes; Full)
 25    Command FIFO Less Than Half Full  (0=No, 1=Yes; Less than Half-full)
 26    Command FIFO Empty                (0=No, 1=Yes; Empty)
 27    Geometry Engine Busy (0=No, 1=Yes; Busy; Commands are executing)
 28-29 Not used
 30-31 Command FIFO IRQ (0=Never, 1=Less than half full, 2=Empty, 3=Reserved)
```

When GXFIFO IRQ is enabled (setting 1 or 2), the IRQ flag (IF.Bit21) is set while and as long as the IRQ condition is true (and attempts to acknowledge the IRQ by writing to IF.Bit21 have no effect). So that, the IRQ handler must either fill the FIFO, or disable the IRQ (setting 0), BEFORE trying to acknowledge the IRQ.

4000604h - RAM_COUNT - Polygon List & Vertex RAM Count Register (R)

```
 0-11   Number of Polygons currently stored in Polygon List RAM (0..2048)
 12-15  Not used
 16-28  Number of Vertices currently stored in Vertex RAM      (0..6144)
 13-15  Not used
```

If a SwapBuffers command has been sent, then the counters are reset 10 cycles (at 33.51MHz clock) after next VBlank.

4000320h - RDLINES_COUNT - Rendered Line Count Register (R)
Rendering starts in scanline 214, the rendered lines are stored in a buffer that can hold up to 48 scanlines. The actual screen output begins after scanline 262, the lines are then read from the buffer and sent to the display. Simultaneously, the rendering engine keeps writing new lines to the buffer (ideally at the same speed than display output, so the buffer would always contain 48 pre-calculated lines).

```
  0-5    Minimum Number (minus 2) of buffered lines in previous frame (0..46)
  6-31   Not used
```
If rendering becomes slower than the display output, then the number of buffered lines decreases. Smaller values in RDLINES indicate that additional load to the rendering engine may cause buffer underflows in further frames, if so, the program should reduce the number of polygons to avoid display glitches.
Even if RDLINES becomes zero, it doesn't indicate whether actual buffer underflows have occured or not (underflows are indicated in DISP3DCNT Bit12).

DS 3D Tests

40005C0h - Cmd 70h - BOX_TEST - Test if Cuboid Sits inside View Volume (W)
The BoxTest result indicates if one or more of the 6 faces of the box are fully or parts of inside of the view volume. Can be used to reduce unnecessary overload, ie. if the result is false, then the program can skip drawing of objects which are inside of the box.
BoxTest verifies only if the faces of the box are inside view volume, and so, it will return false if the whole view volume is located inside of the box (still objects inside of the box may be inside of view).
```
  Parameter 1, Bit 0-15   X-Coordinate
  Parameter 1, Bit 16-31  Y-Coordinate
  Parameter 2, Bit 0-15   Z-Coordinate
  Parameter 2, Bit 16-31  Width  (presumably: X-Offset?)
  Parameter 3, Bit 0-15   Height (presumably: Y-Offset?)
  Parameter 3, Bit 16-31  Depth  (presumably: Z-Offset?)
  All values are 1bit sign, 3bit integer, 12bit fractional part
```
The result of the "coordinate+offset" additions should not overflow 16bit vertex coordinate range (1bit sign, 3bit integer, 12bit fraction).
Before using BoxTest, be sure that far-plane-intersecting & 1-dot polygons are enabled, if they aren't: Send the PolygonAttr command (with bit12,13 set to enable them), followed by dummy Begin and End commands (required to apply the new PolygonAttr settings). BoxTest should not be issued within Begin/End.
After sending the BoxTest command, wait until GXSTAT.Bit0 indicates Ready, then read the result from GXSTAT.Bit1.

40005C4h - Cmd 71h - POS_TEST - Set Position Coordinates for Test (W)
```
  Parameter 1, Bit 0-15   X-Coordinate
  Parameter 1, Bit 16-31  Y-Coordinate
  Parameter 2, Bit 0-15   Z-Coordinate
  Parameter 2, Bit 16-31  Not used
  All values are 1bit sign, 3bit integer, 12bit fractional part.
```
Multiplies the specified line-vector (x,y,z,1) by the clip coordinate matrix.
After sending the command, wait until GXSTAT.Bit0 indicates Ready, then read the result from POS_RESULT registers. POS_TEST can be issued anywhere (except within polygon strips, huh?).
Caution: POS_TEST overwrites the internal VTX registers, so the next vertex should be <fully> defined by VTX_10 or VTX_16, otherwise, when using VTX_XY, VTX_XZ, VTX_YZ, or VTX_DIFF, then the new vertex will be relative to the POS_TEST coordinates (rather than to the previous vertex).

4000620h..62Fh - POS_RESULT - Position Test Results (R)
This 16-byte region (4 words) contains the resulting clip coordinates (x,y,z,w) from the POS_TEST command. Each value is 1bit sign, 19bit integer, 12bit fractional part.

40005C8h - Cmd 72h - VEC_TEST - Set Directional Vector for Test (W)
```
  Parameter 1, Bit 0-9    X-Component
  Parameter 1, Bit 10-19  Y-Component
  Parameter 1, Bit 20-29  Z-Component
  Parameter 1, Bit 30-31  Not used
  All values are 1bit sign, 9bit fractional part.
```
Multiplies the specified line-vector (x,y,z,0) by the directional vector matrix. Similar as for the NORMAL command, it does require Matrix Mode 2 (ie. Position & Vector Simultaneous Set mode).

After sending the command, wait until GXSTAT.Bit0 indicates Ready, then read the result ("the directional vector in the View coordinate space") from VEC_RESULT registers.

4000630h..635h - VEC_RESULT - Vector Test Results (R)
This 6-byte region (3 halfwords) contains the resulting vector (x,y,z) from the VEC_TEST command. Each value is 4bit sign, 0bit integer, 12bit fractional part. The 4bit sign is either 0000b (positive) or 1111b (negative). There is no integer part, so values >=1.0 or <-1.0 will cause overflows.
(Eg. +1.0 aka 1000h will be returned as -1.0 aka F000h due to overflow and sign-expansion).

DS 3D Rear-Plane

Other docs seem to refer to this as Clear-plane, rather than Rear-plane, anyways, the plane can be an image, so it isn't always "cleared".
The view order is as such:

```
--> 2D Layers --> 3D Polygons --> 3D Rear-plane --> 2D Layers --> 2D Backdrop
```

The rear-plane can be disabled (by making it transparent; alpha=0), so that the 2D layers become visible as background.
2D layers can be moved in front of, or behind the 3D layer-group (which is represented as BG0 to the 2D Engine), 2D layers behind BG0 can be used instead of, or additionally to the rear-plane.

The rear-plane can be initialized via below two registers (so all pixels in the plane have the same colors and attributes), this method is used when DISP3DCNT.14 is zero:

4000350h - CLEAR_COLOR - Clear Color Attribute Register (W)

```
0-4    Clear Color, Red
5-9    Clear Color, Green
10-14  Clear Color, Blue
15     Fog (enables Fog to the rear-plane) (doesn't affect Fog of polygons)
16-20  Alpha
21-23  Not used
24-29  Clear Polygon ID (affects edge-marking, at the screen-edges?)
30-31  Not used
```

4000354h - CLEAR_DEPTH - Clear Depth Register (W)

```
0-14   Clear Depth (0..7FFFh) (usually 7FFFh = most distant)
15     Not used
16-31  See Port 4000356h, CLRIMAGE_OFFSET
```

The 15bit Depth is expanded to 24bit as "X=(X*200h)+((X+1)/8000h)*1FFh".

Rear Color/Depth Bitmaps
Alternately, the rear-plane can be initialized by bitmap data (allowing to assign different colors & attributes to each pixel), this method is used when DISP3DCNT.14 is set:
Consists of two bitmaps (one with color data, one with depth data), each containing 256x256 16bit entries, and so, each occupying a whole 128K slot,

```
Rear Color Bitmap (located in Texture Slot 2)
  0-4    Clear Color, Red
  5-9    Clear Color, Green
  10-14  Clear Color, Blue
  15     Alpha (0=Transparent, 1=Solid) (equivalent to 5bit-alpha 0 and 31)
Rear Depth Bitmap (located in Texture Slot 3)
  0-14   Clear Depth, expanded to 24bit as X=(X*200h)+((X+1)/8000h)*1FFh
  15     Clear Fog (Initial fog enable value)
```

This method requires VRAM to be allocated to Texture Slot 2 and 3 (see Memory Control chapter). Of course, in that case the VRAM is used as Rear-plane, and cannot be used for Textures.
The bitmap method is restricted to 1bit alpha values (the register-method allows to use a 5bit alpha value).
The Clear Polygon ID is kept defined in the CLEAR_COLOR register, even in bitmap mode.

4000356h - CLRIMAGE_OFFSET - Rear-plane Bitmap Scroll Offsets (W)
The visible portion of the bitmap is 256x192 pixels (regardless of the viewport setting, which is used only for polygon clipping). Internally, the bitmap is 256x256 pixels, so the bottom-most 64 rows are usually offscreen, unless scrolling is used to move them into view.

```
Bit0-7   X-Offset (0..255; 0=upper row of bitmap)
Bit8-14  Y-Offset (0..255; 0=left column of bitmap)
```

The bitmap wraps to the upper/left edges when exceeding the lower/right edges.

DS 3D Final 2D Output

The final 3D image (consisting of polygons and rear-plane) is passed to 2D Engine A as BG0 layer (provided that DISPCNT is configured to use 3D as BG0).

Scrolling
The BG0HOFS register (4000010h) can be used the scroll the 3D layer horizontally, the scroll region is 512 pixels, consisting of 256 pixels for the 3D image, followed by 256 transparent pixels, and then wrapped to the 3D image again. Vertical scrolling (and rotation/scaling) cannot be used on the 3D layer.

BG Priority Order
The lower 2bit of the BG0CNT register (4000008h) control the priority relative to other BGs and OBJs, so the 3D layer can be in front of or behind 2D layers. All other bits in BG0CNT have no effect on 3D, namely, mosaic cannot be used on the 3D layer.

Special Effects
Special Effects Registers (4000050h..54h) can be used as such:

```
Brightness up/down with BG0 as 1st Target via EVY   (as for 2D)
Blending with BG0 as 2nd Target via EVA/EVB         (as for 2D)
Blending with BG0 as 1st Target via 3D Alpha-values (unlike as for 2D)
```

The latter method probably (?) uses per-pixel 3D alpha values as such: EVA=A/2, and EVB=16-A/2, without using the EVA/EVB settings in 4000052h.

Window Feature
Window Feature (4000040h..4Bh) can be used as for 2D.
"If the 3D screen has highest priority, then alpha-blending is always enabled, regardless of the Window Control register's color effect enable flag [ie. regardless of Bit5 of WIN0IN, WIN1IN, WINOBJ, WINOUT registers]"... not sure if that is true, and if it superseedes the effect selection in Port 4000050h...?

DS Sound

The DS contains 16 hardware sound channels.
The console contains two speakers, arranged left and right of the upper screen, and so, provides stereo sound even without using the headphone socket.

DS Sound Channels 0..15
DS Sound Control Registers
DS Sound Capture
DS Sound Block Diagrams
DS Sound Notes

Power control
When restoring power supply to the sound circuit, do not output any sound during the first 15 milliseconds.

DS Sound Channels 0..15

Each of the 16 sound channels occopies 16 bytes in the I/O region, starting with channel 0 at 4000400h..400040Fh, up to channel 15 at 40004F0h..40004FFh.

40004x0h - NDS7 - SOUNDxCNT - Sound Channel X Control Register (R/W)

```
Bit0-6    Volume Mul   (0..127=silent..loud)
Bit7      Not used     (always zero)
Bit8-9    Volume Div   (0=Normal, 1=Div2, 2=Div4, 3=Div16)
Bit10-14  Not used     (always zero)
Bit15     Hold         (0=Normal, 1=Hold last sample after one-shot sound)
Bit16-22  Panning      (0..127=left..right) (64=half volume on both speakers)
Bit23     Not used     (always zero)
Bit24-26  Wave Duty    (0..7) ;HIGH=(N+1)*12.5%, LOW=(7-N)*12.5% (PSG only)
Bit27-28  Repeat Mode  (0=Manual, 1=Loop Infinite, 2=One-Shot, 3=Prohibited)
Bit29-30  Format       (0=PCM8, 1=PCM16, 2=IMA-ADPCM, 3=PSG/Noise)
Bit31     Start/Status (0=Stop, 1=Start/Busy)
```

All channels support ADPCM/PCM formats, PSG rectangular wave can be used only on channels 8..13, and white noise only on channels 14..15.

40004x4h - NDS7 - SOUNDxSAD - Sound Channel X Data Source Register (W)

```
Bit0-26  Source Address (must be word aligned, bit0-1 are always zero)
Bit27-31 Not used
```

40004x8h - NDS7 - SOUNDxTMR - Sound Channel X Timer Register (W)

```
Bit0-15  Timer Value, Sample frequency, timerval=-(33513982Hz/2)/freq
```

The PSG Duty Cycles are composed of eight "samples", and so, the frequency for Rectangular Wave is 1/8th of the selected sample frequency.
For PSG Noise, the noise frequency is equal to the sample frequency.

40004xAh - NDS7 - SOUNDxPNT - Sound Channel X Loopstart Register (W)

```
Bit0-15  Loop Start, Sample loop start position
         (counted in words, ie. N*4 bytes)
```

40004xCh - NDS7 - SOUNDxLEN - Sound Channel X Length Register (W)

The number of samples for N words is 4*N PCM8 samples, 2*N PCM16 samples, or 8*(N-1) ADPCM samples (the first word containing the ADPCM header). The Sound Length is not used in PSG mode.

```
Bit0-21  Sound length (counted in words, ie. N*4 bytes)
Bit22-31 Not used
```

Minimum length (the sum of PNT+LEN) is 4 words (16 bytes), smaller values (0..3 words) are causing hang-ups (busy bit remains set infinite, but no sound output occurs).

In One-shot mode, the sound length is the sum of (PNT+LEN).
In Looped mode, the length is (1*PNT+Infinite*LEN), ie. the first part (PNT) is played once, the second part (LEN) is repeated infinitely.

DS Sound Control Registers

4000500h - NDS7 - SOUNDCNT - Sound Control Register (R/W)

```
Bit0-6   Master Volume       (0..127=silent..loud)
Bit7     Not used            (always zero)
Bit8-9   Left Output from    (0=Left Mixer, 1=Ch1, 2=Ch3, 3=Ch1+Ch3)
Bit10-11 Right Output from   (0=Right Mixer, 1=Ch1, 2=Ch3, 3=Ch1+Ch3)
Bit12    Output Ch1 to Mixer (0=Yes, 1=No) (both Left/Right)
```

```
  Bit13    Output Ch3 to Mixer (0=Yes, 1=No) (both Left/Right)
  Bit14    Not used            (always zero)
  Bit15    Master Enable       (0=Disable, 1=Enable)
  Bit16-31 Not used            (always zero)
```

4000504h - NDS7 - SOUNDBIAS - Sound Bias Register (R/W)

```
  Bit0-9   Sound Bias    (0..3FFh, usually 200h)
  Bit10-31 Not used      (always zero)
```

After applying the master volume, the signed left/right audio signals are in range -200h..+1FFh (with medium level zero), the Bias value is then added to convert the signed numbers into unsigned values (with medium level 200h).
BIAS output is always enabled, even when Master Enable (SOUNDCNT.15) is off.

The sampling frequency of the mixer is 1.04876 MHz with an amplitude resolution of 24 bits, but the sampling frequency after mixing with PWM modulation is 32.768 kHz with an amplitude resolution of 10 bits.

DS Sound Capture

The DS contains 2 built-in sound capture devices that can capture output waveform data to memory.
Sound capture 0 can capture output from left-mixer or output from channel 0.
Sound capture 1 can capture output from right-mixer or output from channel 2.

4000508h - NDS7 - SNDCAP0CNT - Sound Capture 0 Control Register (R/W)
4000509h - NDS7 - SNDCAP1CNT - Sound Capture 1 Control Register (R/W)

```
  Bit0     Control of Associated Sound Channels (ANDed with Bit7)
            SNDCAP0CNT: Output Sound Channel 1 (0=As such, 1=Add to Channel 0)
            SNDCAP1CNT: Output Sound Channel 3 (0=As such, 1=Add to Channel 2)
            Caution: Addition mode works only if BOTH Bit0 and Bit7 are set.
  Bit1     Capture Source Selection
            SNDCAP0CNT: Capture 0 Source (0=Left Mixer, 1=Channel 0/Bugged)
            SNDCAP1CNT: Capture 1 Source (0=Right Mixer, 1=Channel 2/Bugged)
  Bit2     Capture Repeat        (0=Loop, 1=One-shot)
  Bit3     Capture Format        (0=PCM16, 1=PCM8)
  Bit4-6   Not used              (always zero)
  Bit7     Capture Start/Status  (0=Stop, 1=Start/Busy)
```

4000510h - NDS7 - SNDCAP0DAD - Sound Capture 0 Destination Address (R/W)
4000518h - NDS7 - SNDCAP1DAD - Sound Capture 1 Destination Address (R/W)

```
  Bit0-26  Destination address (word aligned, bit0-1 are always zero)
  Bit27-31 Not used (always zero)
```

Capture start address (also used as re-start address for looped capture).

4000514h - NDS7 - SNDCAP0LEN - Sound Capture 0 Length (W)
400051Ch - NDS7 - SNDCAP1LEN - Sound Capture 1 Length (W)

```
  Bit0-15  Buffer length (1..FFFFh words) (ie. N*4 bytes)
  Bit16-31 Not used
```

Minimum length is 1 word (attempts to use 0 words are interpreted as 1 word).

SOUND1TMR - NDS7 - Sound Channel 1 Timer shared as Capture 0 Timer
SOUND3TMR - NDS7 - Sound Channel 3 Timer shared as Capture 1 Timer
There are no separate capture frequency registers, instead, the sample frequency of Channel 1/3 is shared for Capture 0/1. These channels are intended to output the captured data, so it makes sense that both capture and sound output use the same frequency.

For Capture 0, a=0, b=1, x=0.
For Capture 1, a=2, b=3, x=1.

Capture Bugs

The NDS contains two hardware bugs which do occur when capturing data from ch(a) (SNDCAPxCNT.Bit1=1), if so, either bug occurs depending on whether ch(a)+ch(b) addition is enabled or disabled (SNDCAPxCNT.Bit0).

```
  1) Both Negative Bug - SNDCAPxCNT Bit1=1, Bit0=0 (addition disabled)
   Capture data is accidently set to -8000h if ch(a) and ch(b) are both <0.
   Otherwise the correct capture result is returned, ie. plain ch(a) data,
   not being affected by ch(b) (since addition is disabled).
   Workaround: Ensure that ch(a) and/or ch(b) are >=0 (or disabled).
 2) Overflow Bug - SNDCAPxCNT Bit1=1, Bit0=1 (addition enabled)
   In this mode, Capture data isn't clipped to MinMax(-8000h,+7FFFh),
   instead, it is ANDed with FFFFh, so the sign bit is lost if the
   addition result ch(a)+ch(b) is less/greater than -8000h/+7FFFh.
   Workaround: Reduce ch(a)/ch(b) volume or data to avoid overflows.
```

These bugs occur only for capture (speaker output remains intact), and they occur only when capturing ch(a) (capturing mixer-output works flawless).

ch(a)+ch(b) Channel Addition

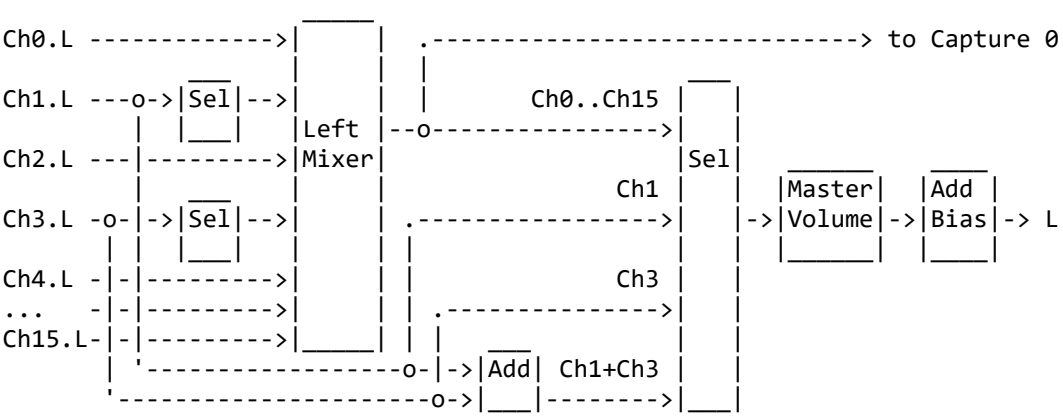
The ch(a)+ch(b) addition unit has 2 outputs, with slightly different results:

```
 1) Addition Result for Capture(x) when using capture source=ch(a):
  Addition is performed always, no matter of SOUNDCNT.Bit12/13.
  And, no matter of ch(a) enable, result is plain ch(b) if ch(a) is disabled.
  Result is 16bit (plus fraction) with overflow error (see Capture Bugs).
 2) Addition Result for Mixer (towards speakers, and capture source=mixer):
  Ch(b) is muted if ch(a) is disabled.
  Ch(b) is muted if ch(b) SOUNDCNT.Bit12/13 is set to "Ch(b) not to mixer".
  Result is 17bit (plus fraction) without overflow error.
```

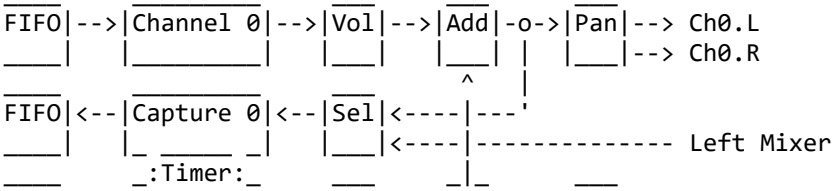
Addition mode can be used only if the <corresponding> capture unit is enabled, ie. if SNDCAPxCNT (Bit0 AND Bit7)=1. If so, addition affects both mixers (and so, may also affect the <other> capture unit if it reads from mixer).

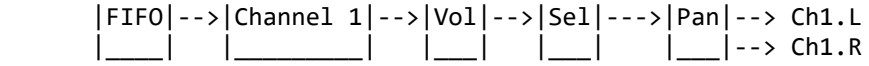
DS Sound Block Diagrams

Left Mixer with Capture 0
(Right Mixer with Capture 1, respectively)

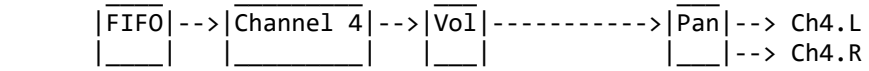


Channel 0 and 1, Capture 0 with input from Left Mixer
(Channel 2 and 3, Capture 1 with input from Right Mixer, respectively)





Channel 4 (Channel 5..15, respectively)



The FIFO isn't used in PSG/Noise modes (supported on channel 8..15).

DS Sound Notes

Sound delayed Start/Restart (timing glitch)
A sound will be started/restarted when changing its start bit from 0 to 1, however, the sound won't start immediately: PSG/Noise starts after 1 sample, PCM starts after 3 samples, and ADPCM starts after 11 samples (3 dummy samples as for PCM, plus 8 dummy samples for the ADPCM header).

Sound Stop (timing note)
In one-shot mode, the Busy bit gets cleared automatically at the BEGIN of the last sample period, nethertheless (despite of the cleared Busy bit) the last sample is kept output until the END of the last sample period (or, if the Hold flag is set, then the last sample is kept output infinitely, that is, until Hold gets cleared, or until the sound gets restarted).

Hold Flag (appears useless/bugged)
The Hold flag allows to keep the last sample being output infinitely after the end of one-shot sounds. This feature is probably intended to allow to play two continous one-shot sound blocks (without producing any scratch noise upon small delays between both blocks, which would occur if the output level would drop to zero). However, the feature doesn't work as intended. As described above, PCM8/PCM16 sound starts are delayed by 3 samples. With Hold flag set, old output level is acually kept intact during the 1st sample, but the output level drops to zero during 2nd-3rd sample, before starting the new sound in 4th sample.

7bit Volume and Panning Values

```
data.vol   = data*N/128
pan.left   = data*(128-N)/128
pan.right  = data*N/128
master.vol = data*N/128/64
```

Register settings of 0..126,127 are interpreted as N=0..126,128.

Max Output Levels
When configured to max volume (and left-most or right-most panning), each channel can span the full 10bit output range (-200h..1FFh) on one speaker, as well as the full 16bit input range (-8000h..7FFFh) on one capture unit.
(It needs 2 channels to span the whole range on BOTH speakers/capture units.)
Together, all sixteen channels could thus reach levels up to -1E00h..21F0h (with default BIAS=200h) on one speaker, and -80000h..+7FFF0h on one capture unit. However, to avoid overflows, speaker outputs are clipped to MinMax(0,3FFh), and capture inputs to MinMax(-8000h..+7FFFh).

Channel/Mixer Bit-Widths

Step	Bits	Min	Max
0 Incoming PCM16 Data	16.0	-8000h	+7FFFh
1 Volume Divider (div 1..16)	16.4	-8000h	+7FFFh
2 Volume Factor (mul N/128)	16.11	-8000h	+7FFFh
3 Panning (mul N/128)	16.18	-8000h	+7FFFh
4 Rounding Down (strip 10bit)	16.8	-8000h	+7FFFh
5 Mixer (add channel 0..15)	20.8	-80000h	+7FFF0h

```
  6 Master Volume (mul N/128/64) 14.21 -2000h     +1FF0h
  7 Strip fraction               14.0  -2000h     +1FF0h
  8 Add Bias (0..3FFh, def=200h) 15.0  -2000h+0   +1FF0h+3FFh
  9 Clip (min/max 0h..3FFh)      10.0  0          +3FFh
```
Table shows integer.fractional bits, and min/max values (without fraction).

Capture Clipping/Rounding
Incoming ch(a) is NOT clipped, ch(a)+ch(b) may overflow (see Capture Bugs).
Incoming mixer data (20.8bits) is clipped to 16.8bits (MinMax -8000h..7FFFh).
For PCM8 capture format, the 16.8 bits are divided by 100h (=8.16 bits).
If the MSB of the fractional part is set, then data is rounded towards zero.
(Positive values are rounded down, negative values are rounded up.)
The fractional part is then discarded, and plain integer data is captured.

PSG Sound
The output volume equals to PCM16 values +7FFFh (HIGH) and -7FFFh (LOW).
PSG sound is always Infinite (the SOUNDxLEN Register, and the SOUNDxCNT Repeat Mode bits have no effect). The PSG hardware doesn't support sound length, sweep, or volume envelopes, however, these effects can be produced by software with little overload (or, more typically, with enormous overload, depending on the programming language used).

PSG Wave Duty (channel 8..13 in PSG mode)
Each duty cycle consists of eight HIGH or LOW samples, so the sound frequency is 1/8th of the selected sample rate. The duty cycle always starts at the begin of the LOW period when the sound gets (re-)started.
```
  0  12.5% "_______-_______-_______-"
  1  25.0% "______--______--______--"
  2  37.5% "_____---_____---_____---"
  3  50.0% "____----____----____----"
  4  62.5% "___-----___-----___-----"
  5  75.0% "__------__------__------"
  6  87.5% "_-------_-------_-------"
  7   0.0% "________________________"
```
The Wave Duty bits exist and are read/write-able on all channels (although they are actually used only in PSG mode on channels 8-13).

PSG Noise (channel 14..15 in PSG mode)
Noise randomly switches between HIGH and LOW samples, the output levels are calculated, at the selected sample rate, as such:
```
  X=X SHR 1, IF carry THEN Out=LOW, X=X XOR 6000h ELSE Out=HIGH
```
The initial value when (re-)starting the sound is X=7FFFh. The formula is more or less same as "15bit polynomial counter" used on 8bit Gameboy and GBA.

PCM8 and PCM16
Signed samples in range -80h..+7Fh (PCM8), or -8000h..+7FFFh (PCM16).
The output volume of PCM8=NNh is equal to PCM16=NN00h.

IMA-ADPCM Format
IMA-ADPCM is a Adaptive Differential Pulse Code Modulation (ADPCM) variant, designed by International Multimedia Association (IMA), the format is used, among others, in IMA-ADPCM compressed Windows .WAV files.
The NDS data consist of a 32bit header, followed by 4bit values (so each byte contains two values, the first value in the lower 4bits, the second in upper 4 bits). The 32bit header contains initial values:
```
  Bit0-15   Initial PCM16 Value (Pcm16bit = -7FFFh..+7FFF) (not -8000h)
  Bit16-22  Initial Table Index Value (Index = 0..88)
  Bit23-31  Not used (zero)
```
In theory, the 4bit values are decoded into PCM16 values, as such:
```
  Diff = ((Data4bit AND 7)*2+1)*AdpcmTable[Index]/8      ;see rounding-error
```

```
  IF (Data4bit AND 8)=0 THEN Pcm16bit = Max(Pcm16bit+Diff,+7FFFh)
  IF (Data4bit AND 8)=8 THEN Pcm16bit = Min(Pcm16bit-Diff,-7FFFh)
  Index = MinMax (Index+IndexTable[Data4bit AND 7],0,88)
```

In practice, the first line works like so (with rounding-error):

```
  Diff = AdpcmTable[Index]/8
  IF (data4bit AND 1) THEN Diff = Diff + AdpcmTable[Index]/4
  IF (data4bit AND 2) THEN Diff = Diff + AdpcmTable[Index]/2
  IF (data4bit AND 4) THEN Diff = Diff + AdpcmTable[Index]/1
```

And, a note on the second/third lines (with clipping-error):

```
  Max(+7FFFh) leaves -8000h unclipped (can happen if initial PCM16 was -8000h)
  Min(-7FFFh) clips -8000h to -7FFFh (possibly unlike windows .WAV files?)
```

Whereas, IndexTable[0..7] = -1,-1,-1,-1,2,4,6,8. And AdpcmTable [0..88] =

```
  0007h,0008h,0009h,000Ah,000Bh,000Ch,000Dh,000Eh,0010h,0011h,0013h,0015h
  0017h,0019h,001Ch,001Fh,0022h,0025h,0029h,002Dh,0032h,0037h,003Ch,0042h
  0049h,0050h,0058h,0061h,006Bh,0076h,0082h,008Fh,009Dh,00ADh,00BEh,00D1h
  00E6h,00FDh,0117h,0133h,0151h,0173h,0198h,01C1h,01EEh,0220h,0256h,0292h
  02D4h,031Ch,036Ch,03C3h,0424h,048Eh,0502h,0583h,0610h,06ABh,0756h,0812h
  08E0h,09C3h,0ABDh,0BD0h,0CFFh,0E4Ch,0FBAh,114Ch,1307h,14EEh,1706h,1954h
  1BDCh,1EA5h,21B6h,2515h,28CAh,2CDFh,315Bh,364Bh,3BB9h,41B2h,4844h,4F7Eh
  5771h,602Fh,69CEh,7462h,7FFFh
```

The closest way to reproduce the AdpcmTable with 32bit integer maths appears:

```
  X=000776d2h, FOR I=0 TO 88, Table[I]=X SHR 16, X=X+(X/10), NEXT I
  Table[3]=000Ah, Table[4]=000Bh, Table[88]=7FFFh, Table[89..127]=0000h
```

When using ADPCM and loops, set the loopstart position to the data part, rather than the header. At the loop end, the SAD value is reloaded to the loop start location, additionally index and pcm16 values are reloaded to the values that have originally appeared at that location. Do not change the ADPCM loop start position during playback.

Microphone Input

For Microphone (and Touchscreen) inputs, see
DS Touch Screen Controller (TSC)

DS System and Built-in Peripherals

DS DMA Transfers
DS Timers
DS Interrupts
DS Maths
DS Inter Process Communication (IPC)
DS Keypad
DS Absent Link Port
DS Real-Time Clock (RTC)
DS Serial Peripheral Interface Bus (SPI)
DS Touch Screen Controller (TSC)
DS Power Management Device
DS Power Control
DS Backwards-compatible GBA-Mode
DS Debug Registers (Emulator/Devkits)

DS DMA Transfers

The DS includes four DMA channels for each CPU (ie. eight channels in total), which are working more or less the same as on GBA:
GBA DMA Transfers

All NDS9 and NDS7 DMA Registers are R/W. The gamepak bit (Bit 27) has been removed (on the NDS9 the bit is used to expand the mode setting to 3bits).

NDS9 DMA
Word count of all channels is expanded to 21bits (max 1..1FFFFFh units, or 0=200000h units), and SAD/DAD registers for all channels support ranges of 0..0FFFFFFEh. The transfer modes (DMACNT Bit27-29) are:

```
0  Start Immediately
1  Start at V-Blank
2  Start at H-Blank (paused during V-Blank)
3  Synchronize to start of display
4  Main memory display
5  DS Cartridge Slot
6  GBA Cartridge Slot
7  Geometry Command FIFO
```

NDS7 DMA
Word Count, SAD, and DAD are R/W, aside from that they do have the same restrictions as on GBA (max 4000h or 10000h units, some addresses limited to 0..07FFFFFEh). DMACNT Bit27 is unused on NDS7. The transfer modes (DMACNT Bit28-29) are:

```
0  Start Immediately
1  Start at V-Blank
2  DS Cartridge Slot
3  DMA0/DMA2: Wireless interrupt, DMA1/DMA3: GBA Cartridge Slot
```

40000E0h - NDS9 only - DMA0FILL - DMA 0 Filldata (R/W)
40000E4h - NDS9 only - DMA1FILL - DMA 1 Filldata (R/W)
40000E8h - NDS9 only - DMA2FILL - DMA 2 Filldata (R/W)
40000ECh - NDS9 only - DMA3FILL - DMA 3 Filldata (R/W)

```
Bit0-31 Filldata
```

The DMA Filldata registers contain 16 bytes of general purpose WRAM, intended to be used as fixed source addresses for DMA memfill operations.
This is useful because DMA cannot read from TCM, and reading from Main RAM would require to recurse cache & write buffer.
The DMA Filldata is used with Src=Fixed and SAD=40000Exh (which isn't optimal because it's doing repeated reads from SAD, and, for that reason, a memfill via STMIA opcodes can be faster than DMA; the DSi's new NDMA channels are providing a faster fill method with Src=Fill and SAD=Unused).

NDS7 Sound DMA
The NDS additionally includes 16 Sound DMA channels, plus 2 Sound Capture DMA channels (see Sound chapter). The priority of these channels is unknown.

NDS9 Cache, Writebuffer, DTCM, and ITCM
Cache and tightly coupled memory are connected directly to the NDS9 CPU, without using the system bus. So that, DMA cannot access DTCM/ITCM, and access to cached memory regions must be handled with care: Drain the writebuffer before DMA-reads, and invalidate the cache after DMA-writes. See,
ARM CP15 System Control Coprocessor
The CPU can be kept running during DMA, provided that it is accessing only TCM (or cached memory), otherwise the CPU is halted until DMA finishes.
Respectively, interrupts executed during DMA will usually halt the CPU (unless the IRQ handler uses only TCM and cache; the IRQ vector at FFFF00xxh must be cached, or relocated to ITCM at 000000xxh, and the IRQ handler may not access IE, IF, or other I/O ports).

NDS Sequential Main Memory DMA
Main RAM has different access time for sequential and non-sequential access. Normally DMA uses sequential access (except for the first word), however, if the source and destination addresses are both in Main RAM, then all accesses become non-sequential. In that case it would be faster to use two DMA transfers, one from Main

RAM to a scratch buffer in WRAM, and one from WRAM to Main RAM.

DS Timers

Same as GBA, except F = 33.513982 MHz (for both NDS9 and NDS7).
GBA Timers
Both NDS9 and NDS7 have four Timers each, eight Timers in total.
The NDS sound controller is having its own frequency generators (unlike GBA, which needed to use Timers to drive channel A/B sounds).

DS Interrupts

4000208h - NDS9/NDS7 - IME - Interrupt Master Enable (R/W)

```
0     Disable all interrupts  (0=Disable All, 1=See IE register)
1-31  Not used
```

4000210h - NDS9/NDS7 - IE - 32bit - Interrupt Enable (R/W)
4000214h - NDS9/NDS7 - IF - 32bit - Interrupt Request Flags (R/W)
Bits in the IE register are 0=Disable, 1=Enable.
Reading IF returns 0=No request, 1=Interrupt Request.
Writing IF acts as 0=No change, 1=Acknowledge (clears that bit).

```
0     LCD V-Blank
1     LCD H-Blank
2     LCD V-Counter Match
3     Timer 0 Overflow
4     Timer 1 Overflow
5     Timer 2 Overflow
6     Timer 3 Overflow
7     NDS7 only: SIO/RCNT/RTC (Real Time Clock)
8     DMA 0
9     DMA 1
10    DMA 2
11    DMA 3
12    Keypad
13    GBA-Slot (external IRQ source) / DSi: None such
14    Not used                       / DSi9: NDS-Slot Card change?
15    Not used                       / DSi: dito for 2nd NDS-Slot?
16    IPC Sync
17    IPC Send FIFO Empty
18    IPC Recv FIFO Not Empty
19    NDS-Slot Game Card Data Transfer Completion
20    NDS-Slot Game Card IREQ_MC
21    NDS9 only: Geometry Command FIFO
22    NDS7 only: Screens unfolding
23    NDS7 only: SPI bus
24    NDS7 only: Wifi    / DSi9: XpertTeak DSP
25    Not used           / DSi9: Camera
26    Not used           / DSi9: Undoc, IF.26 set on FFh-filling 40021Axh
27    Not used           / DSi:  Maybe IREQ_MC for 2nd gamecard?
28    Not used           / DSi: NewDMA0
29    Not used           / DSi: NewDMA1
30    Not used           / DSi: NewDMA2
31    Not used           / DSi: NewDMA3
?     DSi7: any further new IRQs on ARM7 side... in bit13-15,21,25-26?
```

Raw TCM-only IRQs can be processed even during DMA ?
Trying to set all IE bits gives FFFFFFFFh (DSi7) or FFFFFF7Fh (DSi9).

4000218h - DSi7 - IE2 - DSi7 Extra Interrupt Enable Bits
400021Ch - DSi7 - IF2 - DSi7 Extra Interrupt Flags

```
0     DSi7: GPIO18[0]   ;\
1     DSi7: GPIO18[1]   ; maybe 1.8V signals?
2     DSi7: GPIO18[2]   ;/
3     DSi7: Unused (0)
4     DSi7: GPIO33[0] unknown (related to "GPIO330" testpoint on mainboard?)
5     DSi7: GPIO33[1] Headphone connect (HP#SP) (static state)
6     DSi7: GPIO33[2] Powerbutton interrupt (short pulse upon key-down)
7     DSi7: GPIO33[3] sound enable output (ie. not a useful irq-input)
8     DSi7: SD/MMC Controller   ;-Onboard eMMC and External SD Slot
9     DSi7: SD Slot Data1 pin   ;-For SDIO hardware in External SD Slot
10    DSi7: SDIO Controller     ;\Atheros Wifi Unit
11    DSi7: SDIO Data1 pin      ;/
12    DSi7: AES interrupt
13    DSi7: I2C interrupt
14    DSi7: Microphone Extended interrupt
15-31 DSi7: Unused (0)
```

Trying to set all IE2 bits gives 00007FF7h (DSi7) or 00000000h (DSi9).

DTCM+3FFCh - NDS9 - IRQ Handler (hardcoded DTCM address)
380FFFCh - NDS7 - IRQ Handler (hardcoded RAM address)

```
Bit 0-31  Pointer to IRQ Handler
```

NDS7 Handler must use ARM code, NDS9 Handler can be ARM/THUMB (Bit0=Thumb).

DTCM+3FF8h - NDS9 - IRQ Check Bits (hardcoded DTCM address)
380FFF8h - NDS7 - IRQ Check Bits (hardcoded RAM address)

```
Bit 0-31  IRQ Flags (same format as IE/IF registers)
```

When processing & acknowleding interrupts via IF register, the user interrupt handler should also set the corresponding bits of the IRQ Check value (required for BIOS IntrWait and VBlankIntrWait SWI functions).

380FFC0h - DSi7 only - Extra IRQ Check Bits for IE2/IF2 (hardcoded RAM addr)
Same as the above 380FFF8h value, but for new IE2/IF2 registers, intended for use with IntrWait and VBlankIntrWait functions. However, that functions are BUGGED on DSi and won't actually work in practice (they do support only the new 380FFC0h bits, but do accidently ignore the old 380FFF8h bits).

--- Below for other (non-IRQ) exceptions ---

27FFD9Ch - RAM - NDS9 Debug Stacktop / Debug Vector (0=None)
380FFDCh - RAM - NDS7 Debug Stacktop / Debug Vector (0=None)
These addresses contain a 32bit pointer to the Debug Handler, and, memory below of the addresses is used as Debug Stack. The debug handler is called on undefined instruction exceptions, on data/prefetch aborts (caused by the protection unit), on FIQ (possibly caused by hardware debuggers). It is also called by accidental software-jumps to the reset vector, and by unused SWI numbers within range 0..1Fh.

DS Maths

4000280h - NDS9 - DIVCNT - Division Control (R/W)

```
0-1   Division Mode    (0-2=See below) (3=Reserved; same as Mode 1)
2-13  Not used
14    Division by zero (0=Okay, 1=Division by zero error; 64bit Denom=0)
15    Busy             (0=Ready, 1=Busy) (Execution time see below)
16-31 Not used
```

Division Modes and Busy Execution Times

```
Mode  Numer / Denom = Result, Remainder ; Cycles
0     32bit / 32bit = 32bit , 32bit     ; 18 clks
```

```
1     64bit / 32bit = 64bit , 32bit     ; 34 clks
2     64bit / 64bit = 64bit , 64bit     ; 34 clks
```
Division is started when writing to any of the DIVCNT/NUMER/DENOM registers.

4000290h - NDS9 - DIV_NUMER - 64bit Division Numerator (R/W)
4000298h - NDS9 - DIV_DENOM - 64bit Division Denominator (R/W)
Signed 64bit values (or signed 32bit values in 32bit modes, the upper 32bits are then unused, with one exception: the DIV0 flag in DIVCNT is set only if the full 64bit DIV_DENOM value is zero, even in 32bit mode).

40002A0h - NDS9 - DIV_RESULT - 64bit Division Quotient (=Numer/Denom) (R)
40002A8h - NDS9 - DIVREM_RESULT - 64bit Remainder (=Numer MOD Denom) (R)
Signed 64bit values (in 32bit modes, the values are sign-expanded to 64bit).

Division Overflows
Overflows occur on "DIV0" and "-MAX/-1" (eg. -80000000h/-1 in 32bit mode):
```
DIV0     -->  REMAIN=NUMER, RESULT=+/-1 (with sign opposite of NUMER)
-MAX/-1  -->  RESULT=-MAX               (instead +MAX)
```
On overflows in 32bit/32bit=32bit mode: the upper 32bit of the sign-expanded 32bit result are inverted. This feature produces a correct 64bit (+MAX) result in case of the incorrect 32bit (-MAX) result. The feature also applies on DIV0 errors (which makes the sign-expanded 64bit result even more messed-up than the normal 32bit result).
The DIV0 flag in DIVCNT.14 indicates DENOM=0 errors (it does not indicate "-MAX/-1" errors). The DENOM=0 check relies on the full 64bit value (so, in 32bit mode, the flag works only if the unused upper 32bit of DENOM are zero).

40002B0h - NDS9 - SQRTCNT - Square Root Control (R/W)
```
0     Mode (0=32bit input, 1=64bit input)
1-14  Not used
15    Busy (0=Ready, 1=Busy) (Execution time is 13 clks, in either Mode)
16-31 Not used
```
Calculation is started when writing to any of the SQRTCNT/PARAM registers.

40002B4h - NDS9 - SQRT_RESULT - 32bit - Square Root Result (R)
40002B8h - NDS9 - SQRT_PARAM - 64bit - Square Root Parameter Input (R/W)
Unsigned 64bit parameter, and unsigned 32bit result.

IRQ Notes
Push all DIV/SQRT values (parameters and control registers) when using DIV/SQRT registers on interrupt level, and, after restoring them, be sure to wait until the busy flag goes off, before leaving the IRQ handler.

BIOS Notes
The NDS9 and NDS7 BIOSes additionally contain software based division and square root functions, which are NOT using above hardware registers (even the NDS9 functions are raw software).

Timing Notes
The Div/Sqrt timings are counted in 33.51MHz units. Although the calculations are quite fast, mind that reading/writing the result/parameter registers takes up additional clock cycles (especially due to the PENALTY cycle glitch for non-sequential accesses; parts of that problem can be eventually bypassed by using sequential STMIA/LDMIA opcodes) (nethertheless, in some cases, software may be actually faster than the hardware registers; eg. for small 8bit numbers; that of course NOT by using the BIOS software functions which are endless inefficient).

DS Inter Process Communication (IPC)

Allows to exchange status information between ARM7 and ARM9 CPUs.
The register can be accessed simultaneously by both CPUs (without violating access permissions, and without generating waitstates at either side).

4000180h - NDS9/NDS7 - IPCSYNC - IPC Synchronize Register (R/W)

```
  Bit   Dir  Expl.
  0-3   R    Data input from IPCSYNC Bit8-11 of remote CPU (00h..0Fh)
  4-7   -    Not used
  8-11  R/W  Data output to IPCSYNC Bit0-3 of remote CPU   (00h..0Fh)
  12    -    Not used
  13    W    Send IRQ to remote CPU      (0=None, 1=Send IRQ)
  14    R/W  Enable IRQ from remote CPU  (0=Disable, 1=Enable)
  15-31 -    Not used
```

4000184h - NDS9/NDS7 - IPCFIFOCNT - IPC Fifo Control Register (R/W)

```
  Bit   Dir  Expl.
  0     R    Send Fifo Empty Status      (0=Not Empty, 1=Empty)
  1     R    Send Fifo Full Status       (0=Not Full, 1=Full)
  2     R/W  Send Fifo Empty IRQ         (0=Disable, 1=Enable)
  3     W    Send Fifo Clear             (0=Nothing, 1=Flush Send Fifo)
  4-7   -    Not used
  8     R    Receive Fifo Empty          (0=Not Empty, 1=Empty)
  9     R    Receive Fifo Full           (0=Not Full, 1=Full)
  10    R/W  Receive Fifo Not Empty IRQ  (0=Disable, 1=Enable)
  11-13 -    Not used
  14    R/W  Error, Read Empty/Send Full (0=No Error, 1=Error/Acknowledge)
  15    R/W  Enable Send/Receive Fifo    (0=Disable, 1=Enable)
  16-31 -    Not used
```

4000188h - NDS9/NDS7 - IPCFIFOSEND - IPC Send Fifo (W)

```
  Bit0-31  Send Fifo Data (max 16 words; 64bytes)
```

4100000h - NDS9/NDS7 - IPCFIFORECV - IPC Receive Fifo (R)

```
  Bit0-31  Receive Fifo Data (max 16 words; 64bytes)
```

IPCFIFO Notes
When IPCFIFOCNT.15 is disabled: Writes to IPCFIFOSEND are ignored (no data is stored in the FIFO, the error bit doesn't get set though), and reads from IPCFIFORECV return the oldest FIFO word (as usually) (but without removing the word from the FIFO).
When the Receive FIFO is empty: Reading from IPCFIFORECV returns the most recently received word (if any), or ZERO (if there was no data, or if the FIFO was cleared via IPCFIFOCNT.3), and, in either case the error bit gets set.
The Fifo-IRQs are edge triggered, IF.17 gets set when the condition "(IPCFIFOCNT.2 AND IPCFIFOCNT.0)" changes from 0-to-1, and IF.18 gets set when "(IPCFIFOCNT.10 AND NOT IPCFIFOCNT.8)" changes from 0-to-1. The IRQ flags can be acknowledged even while that conditions are true.

DS Keypad

For the GBA-buttons: Same as GBA, both ARM7 and ARM9 have keyboard input registers, and each its own keypad IRQ control register.
GBA Keypad Input

For Touchscreen (and Microphone) inputs, see
DS Touch Screen Controller (TSC)

4000136h - NDS7 - EXTKEYIN - Key X/Y Input (R)

```
  0      Button X     (0=Pressed, 1=Released)
  1      Button Y     (0=Pressed, 1=Released)
  3      DEBUG button (0=Pressed, 1=Released/None such)
  6      Pen down     (0=Pressed, 1=Released/Disabled) (always 0 in DSi mode)
  7      Hinge/folded (0=Open, 1=Closed)
  2,4,5  Unknown / set
  8..15  Unknown / zero
```

The Hinge stuff is a magnetic sensor somewhere underneath of the Start/Select buttons (NDS) or between A/B/X/Y buttons (DSi), it will be triggered by the magnet field from the right speaker when the console is closed. The hinge generates an interrupt request (there seems to be no way to disable this, unlike as for all other IRQ sources), however, the interrupt execution can be disabled in IE register (as for other IRQ sources).
The Pen Down is the /PENIRQ signal from the Touch Screen Controller (TSC), if it is enabled in the TSC control register, then it will notify the program when the screen pressed, the program should then read data from the TSC (if there's no /PENIRQ then doing unneccassary TSC reads would just waste CPU power). However, the user may release the screen before the program performs the TSC read, so treat the screen as not pressed if you get invalid TSC values (even if /PENIRQ was LOW).
Not sure if the TSC /PENIRQ is actually triggering an IRQ in the NDS?
The Debug Button should be connected to R03 and GND (on original NDS, R03 is the large soldering point between the SL1 jumper and the VR1 potentiometer) (there is no R03 signal visible on the NDS-Lite board).
Interrupts are reportedly not supported for X,Y buttons.

DS Absent Link Port

The DS doesn't have a Serial Link Port Socket, however, internally, the NDS7 contains the complete set of Serial I/O Ports, as contained in the GBA:
GBA Communication Ports

In GBA mode, the ports are working as on real GBA (as when no cable is connected). In NDS mode, the ports are even containing some additional bits:

NDS7 SIO Bits (according to an early I/O map from Nintendo)

```
  NDS7 4000128h SIOCNT   Bit15 "CKUP"  New Bit in NORMAL/MULTI/UART mode (R/W)
  NDS7 4000128h SIOCNT   Bit14 "N/A"   Removed IRQ Bit in UART mode (?)
  NDS7 400012Ah SIOCNT_H Bit14 "TFEMP" New Bit (R/W)
  NDS7 400012Ah SIOCNT_H Bit15 "RFFUL" New Bit (always zero?)
  NDS7 400012Ch SIOSEL   Bit0  "SEL"   New Bit (always zero?)
  NDS7 4000140h JOYCNT   Bit7  "MOD"   New Bit (R/W)
```

The "CKUP" bit duplicates the internal clock transfer rate (selected in SIOCNT.1) (tested in normal mode) (probably works also in multi/uart mode?).

NDS7 DS-Lite 4001080h (W) (?)
DS-Lite Firmware writes FFFFh to this address (prior to accessing SIOCNT), so it's probably SIO or debugging related (might be as well a bug or so). Reading from the port always returns 0000h on both DS and DS-Lite.

NDS9 SIO Bits (according to an early I/O map from Nintendo)

```
  NDS9 4000120h SIODATA32 Bit0-31 Data            (always zero?)
  NDS9 4000128h SIOCNT    Bit2    "TRECV" New Bit (always zero?)
  NDS9 4000128h SIOCNT    Bit3    "TSEND" New Bit (always zero?)
  NDS9 400012Ch SIOSEL    Bit0    "SEL"   New Bit (always zero?)
```

Not sure if these ports really exist in the release-version, or if it's been prototype stuff?

RCNT
RCNT (4000134h) should be set to 80xxh (general purpose mode) before accessing EXTKEYIN (4000136h) or

RTC (4000138h). No idea why (except when using RTC/SI-interrupt).

DS Serial Port
The SI line is labeled "INT" on the NDS mainboard, it is connected to Pin 1 of the RTC chip (ie. the /INT interrupt pin).
I have no idea where to find SO, SC, and SD. I've written a test proggy that pulsed all four RCNT bits - but all I could find was the SI signal. However, the BIOS contains some code that uses SIO normal mode transfers (for the debug version), so at least SI, SO, SC should exist...?
MAYBE that three signals are somehow replaced by EXTKEYIN bit0,1,3?

DS Real-Time Clock (RTC)

NDS
Seiko Instruments Inc. S-35180 (compatible with S-35190A)
Miniature 8pin RTC with 3-wire serial bus

DSi
Seiko S-35199A01 (12pin BGA, with some extra functions like FOUT and Alarm Date)

4000138h - NDS7 - Real Time Clock Register

```
Bit  Expl.
0    Data I/O   (0=Low, 1=High)
1    Clock Out  (0=Low, 1=High)
2    Select Out (0=Low, 1=High/Select)
4    Data  Direction  (0=Read, 1=Write)
5    Clock Direction  (should be 1=Write)
6    Select Direction (should be 1=Write)
3,8-11   Unused I/O Lines
7,12-15  Direction for Bit3,8-11 (usually 0)
16-31    Not used
```

Serial Transfer Flowchart
Chipselect and Command/Parameter Sequence:

```
Init CS=LOW and /SCK=HIGH, and wait at least 1us
Switch CS=HIGH, and wait at least 1us
Send the Command byte (see bit-transfer below)
Send/receive Parameter byte(s) associated with the command (see below)
Switch CS to LOW
```

Bit transfer (repeat 8 times per cmd/param byte) (bits transferred LSB first):

```
Output /SCK=LOW and SIO=databit (when writing), then wait at least 5us
Output /SCK=HIGH, wait at least 5us, then read SIO=databit (when reading)
In either direction, data is output on (or immediately after) falling edge.
```

Ideally, <both> commands and parameters should be transmitted LSB-first (unlike the original Seiko document, which recommends LSB-first for data, and MSB-first for commands) (actually, later Seiko datasheets are going so far to recommend MSB-first for everything, eg. to use bit-reversed Data=C8h for Year=13h).

Command Register

```
Command Register
  Fwd  Rev
  0    7   Fixed Code (must be 0)
  1    6   Fixed Code (must be 1)
  2    5   Fixed Code (must be 1)
  3    4   Fixed Code (must be 0, or, DSi only: 1=Extended Command)
  4-6  3-1 Command
           Fwd Rev Parameter bytes (read/write access)
           0   0   1 byte, status register 1
           4   1   1 byte, status register 2
```

```
            2   2   7 bytes, date & time (year,month,day,day_of_week,hh,mm,ss)
            6   3   3 bytes, time (hh,mm,ss)
            1*  4*  1 byte, int1, frequency duty setting
            1*  4*  3 bytes, int1, alarm time 1 (day_of_week, hour, minute)
            5   5   3 bytes, int2, alarm time 2 (day_of_week, hour, minute)
            3   6   1 byte, clock adjustment register
            7   7   1 byte, free register
            Extended command (when above "fourth bit" was set, DSi only)
            Fwd Rev Parameter bytes (read/write access)
            0   0   3 byte, up counter (msw,mid,lsw) (read only)
            4   1   1 byte, FOUT register setting 1
            2   2   1 byte, FOUT register setting 2
            6   3   reserved
            1   4   3 bytes, alarm date 1 (year,month,day)
            5   5   3 bytes, alarm date 2 (year,month,day)
            3   6   reserved
            7   7   reserved
    7    0   Parameter Read/Write Access (0=Write, 1=Read)
```

* INT1: Type and number of parameters depend on INT1 setting in stat reg2.

The "Fwd" bit numbers and command values for LSB-first command transfers (ie. both commands and parameters use the same bit-order).

The "Rev" numbers/values are for MSB-first command transfers (ie. commands using opposite bit-order than parameters, as being suggested by Seiko).

Control and Status Registers

```
  Status Register 1
    0   W   Reset                (0=Normal, 1=Reset)
    1   R/W 12/24 hour mode      (0=12 hour, 1=24 hour)
    2-3 R/W General purpose bits
    4   R   Interrupt 1 Flag (1=Yes)                      ;auto-cleared on read
    5   R   Interrupt 2 Flag (1=Yes)                      ;auto-cleared on read
    6   R   Power Low Flag (0=Normal, 1=Power is/was low) ;auto-cleared on read
    7   R   Power Off Flag (0=Normal, 1=Power was off)    ;auto-cleared on read
    Power off indicates that the battery was removed or fully discharged,
    all registers are reset to 00h (or 01h), and must be re-initialized.
  Status Register 2
    0-3 R/W INT1 Mode/Enable
            0000b Disable
            0x01b Selected Frequency steady interrupt
            0x10b Per-minute edge interrupt
            0011b Per-minute steady interrupt 1 (duty 30.0 seconds)
            0100b Alarm 1 interrupt
            0111b Per-minute steady interrupt 2 (duty 0.0079 seconds)
            1xxxb 32kHz output
    4-5 R/W General purpose bits
    6   R/W INT2 Enable
            0b    Disable
            1b    Alarm 2 interrupt
    7   R/W Test Mode (0=Normal, 1=Test, don't use) (cleared on Reset)
  Clock Adjustment Register (to compensate oscillator inaccuracy)
    0-7 R/W Adjustment (00h=Normal, no adjustment)
  Free Register
    0-7 R/W General purpose bits
```

Date Registers

```
  Year Register
    0-7 R/W Year     (BCD 00h..99h = 2000..2099)
  Month Register
    0-4 R/W Month    (BCD 01h..12h = January..December)
    5-7 -   Not used (always zero)
  Day Register
    0-5 R/W Day      (BCD 01h..28h,29h,30h,31h, range depending on month/year)
    6-7 -   Not used (always zero)
```

```
  Day of Week Register (septenary counter)
    0-2 R/W Day of Week (00h..06h, custom assignment, usually 0=Monday?)
    3-7 -   Not used (always zero)
```

Time Registers

```
  Hour Register
    0-5 R/W Hour     (BCD 00h..23h in 24h mode, or 00h..11h in 12h mode)
    6   *   AM/PM    (0=AM before noon, 1=PM after noon)
            * 24h mode: AM/PM flag is read only (PM=1 if hour = 12h..23h)
            * 12h mode: AM/PM flag is read/write-able
            * 12h mode: Observe that 12 o'clock is defined as 00h (not 12h)
    7   -   Not used (always zero)
  Minute Register
    0-6 R/W Minute   (BCD 00h..59h)
    7   -   Not used (always zero)
  Second Register
    0-6 R/W Minute   (BCD 00h..59h)
    7   -   Not used (always zero)
```

Alarm 1 and Alarm 2 Registers

```
  Alarm1 and Alarm2 Day of Week Registers (INT1 and INT2 each)
    0-2 R/W Day of Week (00h..06h)
    3-6 -   Not used (always zero)
    7   R/W Compare Enable (0=Alarm every day, 1=Alarm only at specified day)
  Alarm1 and Alarm2 Hour Registers (INT1 and INT2 each)
    0-5 R/W Hour     (BCD 00h..23h in 24h mode, or 00h..11h in 12h mode)
    6   R/W AM/PM    (0=AM, 1=PM) (must be correct even in 24h mode?)
    7   R/W Compare Enable (0=Alarm every hour, 1=Alarm only at specified hour)
  Alarm1 and Alarm2 Minute Registers (INT1 and INT2 each)
    0-6 R/W Minute   (BCD 00h..59h)
    7   R/W Compare Enable (0=Alarm every min, 1=Alarm only at specified min)
  Selected Frequency Steady Interrupt Register (INT1 only) (when Stat2/Bit2=0)
    0   R/W Enable 1Hz Frequency  (0=Disable, 1=Enable)
    1   R/W Enable 2Hz Frequency  (0=Disable, 1=Enable)
    2   R/W Enable 4Hz Frequency  (0=Disable, 1=Enable)
    3   R/W Enable 8Hz Frequency  (0=Disable, 1=Enable)
    4   R/W Enable 16Hz Frequency (0=Disable, 1=Enable)
            The signals are ANDed when two or more frequencies are enabled,
            ie. the /INT signal gets LOW when either of the signals is LOW.
    5-7 R/W General purpose bits
```

Note: There is only one register shared as "Selected Frequency Steady Interrupt" (accessed as single byte parameter when Stat2/Bit2=0) and as "Alarm1 Minute" (accessed as 3rd byte of 3-byte parameter when Stat2/Bit2=1), changing either value will also change the other value.

Up Counter (DSi only)

```
  Up Counter Msw
    0-7 R   Up Counter bit16-23 (non-BCD, 00h..FFh)
  Up Counter Mid
    0-7 R   Up Counter bit8-15  (non-BCD, 00h..FFh)
  Up Counter Lsw
    0-7 R   Up Counter bit0-7   (non-BCD, 00h..FFh)
```

The 24bit Up Counter is incremented when seconds=00h (that is, once per minute; unless the Time is getting getting changed by write commands, which may cause some stuttering). The Up Counter starts at 000000h upon power-up, and, if the battery lasts that long: wraps from FFFFFFh to 000000h after about 30 years.

Alarm 1 and Alarm 2 Date Registers (DSi only)

```
  Alarm 1 and Alarm 2 Year Register
    0-7 R/W Year     (BCD 00h..99h = 2000..2099)
  Alarm 1 and Alarm 2 Month Register
    0-4 R/W Month    (BCD 01h..12h = January..December)
    5   -   Not used (always zero)
    6   R/W Year Compare Enable (0=Ignore, 1=Enable)
```

```
  7   R/W Month Compare Enable (0=Ignore, 1=Enable)
Alarm 1 and Alarm 2 Day Register
  0-5 R/W Day      (BCD 01h..28h,29h,30h,31h, range depending on month/year)
  6   -   Not used (always zero)
  7   R/W Day Compare Enable (0=Ignore, 1=Enable)
```
XXX unspecified if above Alarm Date stuff is really R/W (or write only)

FOUT Register (DSi only)
```
FOUT Register Setting 1
  0-7 R/W  Enable bits (bit0=256Hz, bit1=512Hz, ..., bit7=32768Hz)
FOUT Register Setting 2
  0-7 R/W  Enable bits (bit0=1Hz,   bit1=2Hz, ...,   bit7=128Hz)
The above sixteen FOUT signals are ANDed when two or more frequencies are
enabled, ie. the FOUT signal gets LOW when either of the signals is LOW.
```
Note: The FOUT pin goes to the DSi's wifi daughterboard (FOUT is configured by firmware (needed if it was changed, or when the battery was removed), FOUT is required for exchanging Atheros WMI commands/events).

Interrupt
There's only one /INT signal, shared for both INT1 and INT2.
In the NDS, it is connected to the SI-input of the SIO unit (and so, also shared with SIO interrupts). To enable the interrupt, RCNT should be set to 8144h (Bit14-15=General Purpose mode, Bit8=SI Interrupt Enable, Bit6,2=SI Output/High).
The Output/High settings seems to be used as pullup (giving faster reactions on low-to-high transitions) (nethertheless, in most cases it seems to be also working okay as Input, ie. with RCNT=8100h).
The RCNT interrupt is generated on high-to-low transitions on the SI line (but only if the IRQ is enabled in RCNT.8, and only if RCNT is set to general purpose mode) (note: changing RCNT.8 from off-to-on does NOT generate IRQs, even when SI is LOW).

Pin-Outs
```
1 /INT      8 VDD
2 XOUT      7 SIO
3 XIN       6 /SCK
4 GND       5 CS
```

DS Serial Peripheral Interface Bus (SPI)

Serial Peripheral Interface Bus
SPI Bus is a 4-wire (Data In, Data Out, Clock, and Chipselect) serial bus.
The NDS supports the following SPI devices (each with its own chipselect).
DS Firmware Serial Flash Memory
DS Touch Screen Controller (TSC)
DS Power Management Device

40001C0h - NDS7 - SPICNT - SPI Bus Control/Status Register
```
0-1   Baudrate (0=4MHz/Firmware, 1=2MHz/Touchscr, 2=1MHz/Powerman., 3=512KHz)
2     DSi: Baudrate MSB   (4=8MHz, 5..7=None/0Hz) (when SCFG_EXT7.bit9=1)
2     NDS: Not used       (Zero)
3-6   Not used           (Zero)
7     Busy Flag           (0=Ready, 1=Busy) (presumably Read-only)
8-9   Device Select       (0=Powerman., 1=Firmware, 2=Touchscr, 3=Reserved)
10    Transfer Size       (0=8bit/Normal, 1=16bit/Bugged)
11    Chipselect Hold     (0=Deselect after transfer, 1=Keep selected)
12-13 Not used            (Zero)
14    Interrupt Request   (0=Disable, 1=Enable)
15    SPI Bus Enable      (0=Disable, 1=Enable)
```
The "Hold" flag should be cleared BEFORE transferring the LAST data unit, the chipselect will be then automatically cleared after the transfer, the program should issue a WaitByLoop(3) manually AFTER the LAST

transfer.

40001C2h - NDS7 - SPIDATA - SPI Bus Data/Strobe Register (R/W)
The SPI transfer is started on writing to this register, so one must <write> a dummy value (should be zero) even when intending to <read> from SPI bus.

```
0-7   Data
8-15  Not used (always zero, even in bugged-16bit mode)
```

During transfer, the Busy flag in SPICNT is set, and the written SPIDATA value is transferred to the device (via output line), simultaneously data is received (via input line). Upon transfer completion, the Busy flag goes off (with optional IRQ), and the received value can be then read from SPIDATA, if desired.

Notes/Glitches
SPICNT Bits 12,13 appear to be unused (always zero), although the BIOS (attempts to) set Bit13=1, and Bit12=Bit11 when accessing the firmware.
The SPIDATA register is restricted to 8bit, so that only each 2nd byte will appear in SPIDATA when attempting to use the bugged-16bit mode.

Cartridge Backup Auxiliar SPI Bus
The NDS Cartridge Slot uses a separate SPI bus (with other I/O Ports), see
DS Cartridge Backup

DS Touch Screen Controller (TSC)

Texas Instruments TSC2046 (NDS)
Asahi Kasei Microsystems AK4148AVT (NDS-Lite)
The Touch Screen Controller (for lower LCD screen) is accessed via SPI bus,
DS Serial Peripheral Interface Bus (SPI)

Control Byte (transferred MSB first)

```
0-1  Power Down Mode Select
2    Reference Select (0=Differential, 1=Single-Ended)
3    Conversion Mode  (0=12bit, max CLK=2MHz, 1=8bit, max CLK=3MHz)
4-6  Channel Select   (0-7, see below)
7    Start Bit (Must be set to access Control Byte)
```

Channel

```
0 Temperature 0 (requires calibration, step 2.1mV per 1'C accuracy)
1 Touchscreen Y-Position  (somewhat 0B0h..F20h, or FFFh=released)
2 Battery Voltage         (not used, connected to GND in NDS, always 000h)
3 Touchscreen Z1-Position (diagonal position for pressure measurement)
4 Touchscreen Z2-Position (diagonal position for pressure measurement)
5 Touchscreen X-Position  (somewhat 100h..ED0h, or 000h=released)
6 AUX Input               (connected to Microphone in the NDS)
7 Temperature 1 (difference to Temp 0, without calibration, 2'C accuracy)
```

All channels can be accessed in Single-Ended mode.
In differential mode, only channel 1,3,4,5 (X,Z1,Z2,Y) can be accessed.
On AK4148AVT, channel 6 (AUX) is split into two separate channels, IN1 and IN2, separated by Bit2 (Reference Select). IN1 is selected when Bit2=1, IN2 is selected when Bit2=0 (despite of the Bit2 settings, both IN1 and IN2 are using single ended more). On the NDS-Lite, IN1 connects to the mircrophone (as on original NDS), and the new IN2 input is simply wired to VDD3.3 (which is equal to the external VREF voltage, so IN2 is always FFFh).

Power Down Mode

```
Mode /PENIRQ   VREF  ADC   Recommended use
0    Enabled   Auto  Auto  Differential Mode (Touchscreen, Penirq)
```

```
  1    Disabled  Off   On    Single-Ended Mode (Temperature, Microphone)
  2    Enabled   On    Off   Don't use
  3    Disabled  On    On    Don't use
```
Allows to enable/disable the /PENIRQ output, the internal reference voltage (VREF), and the Analogue-Digital Converter.
For AK4148AVT, Power Down modes are slightly different (among others, /PENIRQ is enabled in Mode 0..2).

Reference Voltage (VREF)
VREF is used as reference voltage in single ended mode, at 12bit resolution one ADC step equals to VREF/4096. The TSC generates an internal VREF of 2.5V (+/-0.05V), however, the NDS uses as external VREF of 3.33V (sinks to 3.31V at low battery charge), the external VREF is always enabled, no matter if internal VREF is on or off. Power Down Mode 1 disables the internal VREF, which may reduce power consumption in single ended mode. After conversion, Power Down Mode 0 should be restored to re-enable the Penirq signal.

Sending the first Command after Chip-Select
Switch chipselect low, then output the command byte (MSB first).

Reply Data
The following reply data is received (via Input line) after the Command byte has been transferred: One dummy bit (zero), followed by the 8bit or 12bit conversion result (MSB first), followed by endless padding (zero).
Note: The returned ADC value may become unreliable if there are longer delays between sending the command, and receiving the reply byte(s).

Sending further Commands during/after receiving Reply Data
In general, the Output line should be LOW during the reply period, however, once when Data bit6 has been received (or anytime later), a new Command can be invoked (started by sending the HIGH-startbit, ie. Command bit7), simultaneously, the remaining reply-data bits (bit5..0) can be received.
In other words, the new command can be output after receiving 3 bits in 8bit mode (the dummy bit, and data bits 7..6), or after receiving 7 bits in 12bit mode (the dummy bit, and data bits 11..6).
In practice, the NDS SPI register always transfers 8 bits at once, so that one would usually receive 8 bits (rather than above 3 or 7 bits), before outputting a new command.

Touchscreen Position
Read the X and Y positions in 12bit differential mode, then convert the touchscreen values (adc) to screen/pixel positions (scr), as such:
```
  scr.x = (adc.x-adc.x1) * (scr.x2-scr.x1) / (adc.x2-adc.x1) + (scr.x1-1)
  scr.y = (adc.y-adc.y1) * (scr.y2-scr.y1) / (adc.y2-adc.y1) + (scr.y1-1)
```
The X1,Y1 and X2,Y2 calibration points are found in Firmware User Settings,
DS Firmware User Settings
scr.x1,y1,x2,y2 are originated at 1,1 (converted to 0,0 by above formula).

Touchscreen Pressure (not supported on DSi)
To calculate the pressure resistance, in respect to X/Y/Z positions and X/Y plate resistances, either of below formulas can be used,
```
  Rtouch = (Rx_plate*Xpos*(Z2pos/Z1pos-1))/4096
  Rtouch = (Rx_plate*Xpos*(4096/Z1pos-1)-Ry_plate*(1-Ypos))/4096
```
The second formula requires less CPU load (as it doesn't require to measure Z2), the downside is that one must know both X and Y plate resistance (or at least their ratio). The first formula doesn't require that ratio, and so Rx_plate can be set to any value, setting it to 4096 results in
```
  touchval = Xpos*(Z2pos/Z1pos-1)
```
Of course, in that case, touchval is just a number, not a resistance in Ohms.

Touchscreen Notes
It may be impossible to press locations close to the screen borders.

When pressing two or more locations the TSC values will be somewhere in the middle of these locations.
The TSC values may be garbage if the screen becomes newly pressed or released, to avoid invalid inputs: read TSC values at least two times, and ignore BOTH positions if ONE position was invalid.

Microphone / AUX Channel
Observe that the microphone amplifier is switched off after power up, see:
DS Power Management Device
DS Power Control

Temperature Calculation (not supported on DSi)
TP0 decreases by circa 2.1mV per degree Kelvin. The voltage difference between TP1 minus TP0 increases by circa 0.39mV (1/2573 V) per degree Kelvin. At VREF=3.33V, one 12bit ADC step equals to circa 0.8mV (VREF/4096).
Temperature can be calculated at best resolution when using the current TP0 value, and two calibration values (an ADC value, and the corresponding temperature in degrees kelvin):

```
K = (CAL.TP0-ADC.TP0) * 0.4 + CAL.KELVIN
```

Alternately, temperature can be calculated at rather bad resolution, but without calibration, by using the difference between TP1 and TP0:

```
K = (ADC.TP1-ADC.TP0) * 8568 / 4096
```

To convert Kelvin to other formats,

```
Celsius:     C = (K-273.15)
Fahrenheit:  F = (K-273.15)*9/5+32
Reaumur:     R = (K-273.15)*4/5
Rankine:     X = (K)*9/5
```

The Temperature Range for the TSC 2046 chip is -40'C..+85'C (for AK4181AVT only -20'C..+70'C). According to Nintendo, the DS should not be exposed to "extreme" heat or cold, the optimal battery charging temperature is specified as +10'C..+40'C.
The original firmware does not support temperature calibration, calibration is supported by nocash firmware (if present). See Extended Settings,
DS Firmware Extended Settings

Pin-Outs

```
          ________
VCC  1|o       |16 DCLK
X+   2|        |15 /CS
Y+   3|  TSC   |14 DIN
X-   4|  2046  |13 BUSY
Y-   5|        |12 DOUT
GND  6|        |11 /PENIRQ
VBAT 7|        |10 IOVDD
AUX  8|________|9  VREF
```

For AK4181AVT, same pins as above, except that IOVDD replaced by the new IN2 input, the pin is wired to VDD3.3 (so IN2 is always equal to VREF, which is wired to VDD3.3, too) (and AUX is renamed to IN1, and is kept used for MIC input).

DSi Touchscreen Controller (in NDS mode)
DSi in NDS mode does support only X, Y, and MIC (all other channels do return FFFh in 12bit mode, and FFh in 8bit mode, ie. no pressure, no temperature, and no GNDed battery sensor). On DSi, MIC does return data in both single-ended and differential mode (unlike as on real NDS).

DSi Touchscreen Controller (in DSi mode)
The DSi touchscreen controller supports a NDS backwards compatibility mode. But, in DSi mode, it is working entirely different (it's still accessed via SPI bus, but with some new MODE/INDEX values).
DSi Touchscreen/Sound Controller
The NDS Touchscreen controller did additionally allow to read Temperature and Touchscreen Pressure -

unknown if the DSi is also supporting such stuff (via whatever DSi-specific registers).
The touchscreen hardware can be switched to NDS compatibility mode (for older games), but unknown how to do that.

DS Power Control

The DS contains several Power Managment functions, some accessed via I/O ports (described below), and some accessed via SPI bus:
DS Power Management Device

4000304h - NDS9 - POWCNT1 - Graphics Power Control Register (R/W)

```
0     Enable Flag for both LCDs (0=Disable) (Prohibited, see notes)
1     2D Graphics Engine A      (0=Disable) (Ports 008h-05Fh, Pal 5000000h)
2     3D Rendering Engine       (0=Disable) (Ports 320h-3FFh)
3     3D Geometry Engine        (0=Disable) (Ports 400h-6FFh)
4-8   Not used
9     2D Graphics Engine B      (0=Disable) (Ports 1008h-105Fh, Pal 5000400h)
10-14 Not used
15    Display Swap (0=Send Display A to Lower Screen, 1=To Upper Screen)
16-31 Not used
```

Use SwapBuffers command once after enabling Rendering/Geometry Engine.
Improper use of Bit0 may damage the hardware?
When disabled, corresponding Ports become Read-only, corresponding (palette-) memory becomes read-only-zero-filled.

4000304h - NDS7 - POWCNT2 - Sound/Wifi Power Control Register (R/W)

```
Bit   Expl.
0     Sound Speakers (0=Disable, 1=Enable) (Initial setting = 1)
1     Wifi           (0=Disable, 1=Enable) (Initial setting = 0)
2-31  Not used
```

Note: Bit0 disables the internal Speaker only, headphones are not disabled.
Bit1 disables Port 4000206h, and Ports 4800000h-480FFFFh.

4000206h - NDS7 - WIFIWAITCNT - Wifi Waitstate Control

```
Bit   Expl.
0-2   Wifi WS0 Control (0-7) (Ports 4800000h-4807FFFh)
3-5   Wifi WS1 Control (0-7) (Ports 4808000h-480FFFFh)
4-15  Not used (zero)
```

This register is initialized by firmware on power-up, don't change.
Note: WIFIWAITCNT can be accessed only when enabled in POWCNT2.

4000301h - NDS7 - HALTCNT - Low Power Mode Control (R/W)

In Halt mode, the CPU is paused as long as (IE AND IF)=0.
In Sleep mode, most of the hardware including sound and video are paused, this very-low-power mode could be used much like a screensaver.

```
Bit   Expl.
0-5   Not used (zero)
6-7   Power Down Mode  (0=No function, 1=Enter GBA Mode, 2=Halt, 3=Sleep)
```

The HALTCNT register should not be accessed directly. Instead, the BIOS Halt, Sleep, CustomHalt, IntrWait, or VBlankIntrWait SWI functions should be used.
BIOS Halt Functions
ARM CP15 System Control Coprocessor
The NDS9 does not have a HALTCNT register, instead, the Halt function uses the co-processor opcode "mcr p15,0,r0,c7,c0,4" - this opcode locks up if interrupts are disabled via IME=0 (unlike NDS7 HALTCNT method which doesn't check IME).

4000300h - NDS7/NDS9 - POSTFLG - BYTE - Post Boot Flag (R/W)
The NDS7 and NDS9 post boot flags are usually set upon BIOS/Firmware boot completion, once when set the reset vector is redirected to the debug handler of Nintendo's hardware debugger. That allows the NDS7 debugger to capture accidental jumps to address 0, that appears to be a common problem with HLL-programmers, asm-coders know that (and why) they should not jump to 0.

```
Bit   Expl.
0     Post Boot Flag (0=Boot in progress, 1=Boot completed)
1     NDS7: Not used (always zero), NDS9: Bit1 is read-writeable
2-7   Not used (always zero)
```

There are some write-restrictions: The NDS7 register can be written to only from code executed in BIOS (done by NDS boot ROM, or by DSi firmware, whereas the DSi firmware is using the CpuSet SWI function to issue the POSTFLG write from within ROM). Bit0 of both NDS7 and NDS9 registers cannot be cleared (except by Reset) once when it is set. DSi games seem to run regardless of POSTFLG, whilst NDS games somewhat refuse to run when POSTFLG=0.

Memory Power Down Functions
DS Main Memory Control
DS Firmware Serial Flash Memory

DS Power Management Device

Power Management Device - Mitsumi 3152A (NDS) / Mitsumi 3205B (NDS-LITE)
The Power Management Device is accessed via SPI bus,
DS Serial Peripheral Interface Bus (SPI)
To access the device, write the Index Register, then read or write the data register, and release the chipselect line when finished.

```
Index Register
Bit0-6 Register Select          (0..3) (0..4 for DS-Lite) (0..7Fh for DSi)
Bit7   Register Direction       (0=Write, 1=Read)
Register 0 - Powermanagement Control (R/W)
Bit0   Sound Amplifier Enable   (0=Disable, 1=Enable)
       (Old-DS:  Disabled: Sound is very silent, but still audible)
       (DS-Lite: Disabled: Sound is NOT audible)
       (DSi in NDS Mode: R/W, but effect is unknown yet)
       (DSi in DSi Mode: Not used, Bit0 is always 1)
Bit1   Sound Amplifier Mute     (0=Normal, 1=Mute) (Old-DS Only, not DS-Lite)
       (Old-DS:  Muted: Sound is NOT audible, that works only if Bit0=1)
       (DS-Lite: Not used, Bit1 is always zero)
       (DSi in NDS Mode: R/W, but effect is unknown yet)
       (DSi in DSi Mode: R/W, but effect is unknown yet)
Bit2   Lower Backlight          (0=Disable, 1=Enable)
Bit3   Upper Backlight          (0=Disable, 1=Enable)
Bit4   Power LED Blink Enable   (0=Always ON, 1=Blinking OFF/ON)
Bit5   Power LED Blink Speed    (0=Slow, 1=Fast) (only if Blink enabled)
       (DSi: Power LED Blinking isn't supported, neither in NDS nor DSi mode)
Bit6   DS System Power          (0=Normal, 1=Shut Down)
Bit7   Not used                 (always 0)
Register 1 - Battery Status (R)
Bit0   Battery Power LED Status (0=Power Good/Green, 1=Power Low/Red)
       (DSi: Usually 0, not tested if it changes upon Power=Low)
Bit1-7 Not used
Register 2 - Microphone Amplifier Control (R/W)
Bit0   Amplifier                (0=Disable, 1=Enable)
Bit1-7 Not used                 (always 0)
(DSi in NDS Mode: looks same as NDS, ie. only bit0 is R/W)
(DSi in DSi Mode: Not used, always FFh)
Register 3 - Microphone Amplifier Gain Control (R/W)
Bit0-1 Gain                     (0..3=Gain 20, 40, 80, 160)
Bit2-7 Not used                 (always 0)
```

```
(DSi in NDS Mode: looks same as NDS, ie. only bit0-1 are R/W)
(DSi in DSi Mode: Not used, always FFh)
Register 4 - DS-Lite and DSi Only - Backlight Levels/Power Source (R/W)
Bit0-1 Backlight Brightness (0..3=Low,Med,High,Max)   (R/W)
       (when bit2+3 are both set, then reading bit0-1 always returns 3)
Bit2   Force Max Brightness when Bit3=1 (0=No, 1=Yes) (R/W)
Bit3   External Power Present           (0=No, 1=Yes) (Read-Only)
Bit4-7 Unknown (Always 4) (Read-Only)
(DSi in NDS Mode: looks same as in DSi mode)
(DSi in DSi Mode: Bit0-1 are R/W, but ignored, bit2-3 are always 0)
Register 10h - DSi Only - Backlight Mirrors & Reset (R/W)
Bit0   Reset (0=No, 1=Reboot DSi) (same/similar as BPTWL reset feature?)
Bit1   Unknown (R/W) (note: whatever it is, it isn't warmboot flag)
Bit2-3 Mirror of Register 0, bit2-3 (backlight enable bits) (R/W)
Bit4-7 Not used (always 0)
(This register works in NDS mode and DSi mode, though it's mainly intended
for NDS mode, eg. DS Download Play uses the Reset bit to return to DSi menu)
(note: writing bit2 seems to affect BOTH bit1 and bit2 in register 0)
```

On Old-DS, registers 4..7Fh are mirrors of 0..3. On DS-Lite, registers 5,6,7 are mirrors of 4, register 8..7Fh are mirrors of 0-7.
On DSi (in DS mode), index 0,1,2,3,4,10h are used (reads as 0Fh,00h,00h,01h,41h,0Fh - regardless of backlight level, and power source), index 5..0Fh and 11h..7Fh return 00h (ie. unlike DS and DS-Lite, there are no mirrors; aside from the mirrored bits in register 10h).

Backlight Dimming / Backlight caused Shut-Down(s)
The above bits are essentially used to switch Backlights on or off. However, there a number of strange effects. Backlight dimming is possible by pulse width modulation, ie. by using a timer interrupt to issue pulse widths of N% ON, and 100-N% OFF. Too long pulses are certainly resulting in flickering. Too short pulses are ignored, the backlights will remain OFF, even if the ON and OFF pulses are having the same length. Much too short pulses cause the power supply to shut-down; after changing the backlight state, further changes must not occur within the next (circa) 2500 clock cycles. The mainboard can be operated without screens & backlights connected, however, if so, the power supply will shut-down as soon as backlights are enabled.
Pulse width modulated dimming does also work on the DS-Lite, allowing to use smoother fade in/out effects as when using the five "hardware" levels (Off,Low,Med,High,Max).

DS Main Memory Control

Main Memory
The DS Main Memory is 2Mx16bit (4MByte), 1.8V Pseudo SRAM (PSRAM); all Dynamic RAM refresh is handled internally, the chip doesn't require any external refresh signals, and alltogether behaves like Static RAM. Non-sequential access time is 70ns, sequential (burst) access time is 12ns.

Main Memory Control
The memory chips contain built-in Control functions, which can be accessed via Port 27FFFFEh and/or by EXMEMCNT Bit 14. Nintendo is using at least two different types of memory chips in DS consoles, Fujitsu 82DBS02163C-70L, and ST M69AB048BL70ZA8, both appear to have different control mechanisms, other chips (with 8MB size) are used in the semi-professional DS hardware debuggers, and further chips may be used in future, so using the memory control functions may lead into compatibitly problems.

Power Consumption / Power Control
Power Consumption during operation (read/write access) is somewhat 30mA, in standby mode (no read/write access) consumption is reduced to 100uA.
Furthermore, a number of power-down modes are supported: In "Deep" Power Down mode the refresh is fully disabled, consumption is 10uA (and all data will be lost), in "Partial" Power Down modes only fragment of memory is refreshed, for smallest fragments, consumption goes to down to circa 50uA. The chip cannot be

accessed while it is in Deep or Partial Power Down mode.

Fujitsu 82DBS02163C-70L
The Configuration Register (CR) can be written to by the following sequence:

```
LDRH R0,[27FFFFEh]      ;read one value
STRH R0,[27FFFFEh]      ;write should be same value as above
STRH R0,[27FFFFEh]      ;write should be same value as above
STRH R0,[27FFFFEh]      ;write any value
STRH R0,[27FFFFEh]      ;write any value
LDRH R0,[2400000h+CR*2] ;read, address-bits are defining new CR value
```

Do not access any other Main Memory addresses during above sequence (ie. disable interrupts, and do not execute the sequence by code located in Main Memory). The CR value is write-only. The CR bits are:

```
Bit    Expl.
0-6    Reserved         (Must be 7Fh)
7      Write Control
         0=WE Single Clock Pulse Control without Write Suspend Function
         1=WE Level Control with Write Suspend Function)
        Burst Read/Single Write is not supported at WE Single Clock Mode.
8      Reserved         (Must be 1)
9      Valid Clock Edge (0=Falling Edge, 1=Rising Edge)
10     Single Write     (0=Burst Read/Burst Write, 1=Burst Read/Single Write)
11     Burst Sequence   (0=Reserved, 1=Sequential)
12-14  Read Latency     (1=3 clocks, 2=4 clocks, 3=5 clocks, other=Reserved)
15     Mode
         0=Synchronous:  Burst Read, Burst Write
         1=Asynchronous: Page Read, Normal Write
        In Mode 1 (Async), only the Partial Size bits are used,
        all other bits, CR bits 0..18, must be "1".
16-18  Burst Length     (2=8 Words, 3=16Words, 7=Continous, other=Reserved)
19-20  Partial Size     (0=1MB, 1=512KB, 2=Reserved, 3=Deep/0 bytes)
```

The Power Down mode is entered by setting CE2=LOW, this can be probably done by setting EXMEMCNT Bit14 to zero.

ST Microelectronics M69AB048BL70ZA8
The chip name decodes as PSRAM (M96), Asynchronous (A), 1.8V Burst (B), 2Mx16 (048), Two Chip Enables (B), Low Leakage (L), 70ns (70), Package (ZA), -30..+85'C (8).
There are three data sheets for different PSRAM chips available at www.st.com (unfortunately none for M69AB048BL70ZA8), each using different memory control mechanisms.

NDS9 BIOS
The NDS9 BIOS contains the following Main Memory initialization code, that method doesn't match up with any ST (nor Fujitsu) data sheets that I've seen. At its best, it looks like a strange (and presumably non-functional) mix-up of different ST control methods.

```
STRH 2000h,[4000204h]    ;EXMEMCNT, enable RAM, async mode
LDRH R0,[27FFFFEh]
STRH R0,[27FFFFEh]
STRH R0,[27FFFFEh]
STRH FFDFh,[27FFFFEh]
STRH E732h,[27FFFFEh]
LDRH R0,[27E57FEh]
STRH 6000h,[4000204h]    ;EXMEMCNT, enable RAM, normal mode
```

DS Backwards-compatible GBA-Mode

When booting a 32pin GBA cartridge, the NDS is automatically switched into GBA mode, in that mode all NDS related features are disabled, and the console behaves (almost) like a GBA.

GBA Features that are NOT supported on NDS in GBA Mode.

Unlike real GBA, the NDS does not support 8bit DMG/CGB cartridges.
The undocumented Internal Memory Control register (Port 800h) isn't supported, so the NDS doesn't allow to use 'overclocked' RAM.
The NDS doesn't have a link-port, so GBA games can be played only in single player mode, link-port accessories cannot be used, and the NDS cannot run GBA code via multiboot.

GBA Features that are slightly different on NDS in GBA Mode.
The CPU, Timers, and Sound Frequencies are probably clocked at 16.76MHz; 33.51MHz/2; a bit slower than the original GBA's 16.78MHz clock?
In the BIOS, a single byte in a formerly 00h-filled area has been changed from 00h to 01h, resulting in SWI 0Dh returning a different BIOS checksum.
The GBA picture can be shown on upper or lower screen (selectable in boot-menu), the backlight for the selected screen is always on, resulting in different colors & much better visibility than original GBA. Unlike GBA-SP, the NDS doesn't have a backlight-button.

Screen Border in GBA mode
The GBA screen is centered in the middle of the NDS screen. The surrounding pixels are defined by 32K-color bitmap data in VRAM Block A and B. Each frame, the GBA picture is captured into one block, and is displayed in the next frame (while capturing new data to the other block).
To get a flicker-free border, both blocks should be initialized to contain the same image before entering GBA mode (usually both are zero-filled, resulting in a plain black border).
Note: When using two different borders, the flickering will be irregular - so there appears to be a frame inserted or skipped once every some seconds in GBA mode?!

Switching from NDS Mode to GBA Mode
```
--- NDS9: ---
ZEROFILL VRAM A,B     ;init black screen border (or other color/image)
POWCNT=8003h          ;enable 2D engine A on upper screen (0003h=lower)
EXMEMCNT=...          ;set Async Main Memory mode (clear bit14)
IME=0                 ;disable interrupts
SWI 06h               ;halt with interrupts disabled (lockdown)
--- NDS7: ---
POWERMAN.REG0=09h     ;enable sound amplifier & upper backlight (05h=lower)
IME=0                 ;disable interrupts
wait for VCOUNT=200   ;wait until VBlank
SWI 1Fh with R2=40h   ;enter GBA mode, by CustomHalt(40h)
```
After that, the GBA BIOS will be booted, the GBA Intro will be displayed, and the GBA cartridge (if any) will be started.

DS Debug Registers (Emulator/Devkits)

No$gba Emulator Pseudo I/O Ports (no$gba) (GBA,NDS9,NDS7)
```
4FFFA00h..A0Fh R Emulation ID (16 bytes, eg. "no$gba v2.7", padded with 20h)
4FFFA10h       W String Out (raw)
4FFFA14h       W String Out (with %param's)
4FFFA18h       W String Out (with %param's, plus linefeed)
4FFFA1Ch       W Char Out (nocash)
4FFFA20h..A27h R Clock Cycles (64bit)
4FFFA28h..A3Fh - N/A
```
Note: Above ports can be disabled via the "Debug I/O" option in no$gba setup.

Ensata Emulator Pseudo I/O Ports (NDS9)
```
4000640h (32bit) ;aka CLIPMTX_RESULT (mis-used to invoke detection)
4000006h (16bit) ;aka VCOUNT (mis-used to get detection result)
4FFF010h (32bit) ;use to initialize/unlock/reset something
4FFF000h (8bit)  ;debug message character output (used when Ensata detected)
```

The Ensata detection works by mis-using CLIPMTX_RESULT and VCOUNT registers:

```
  [4000640h]=2468ACE0h      ;CLIPMTX_RESULT (on real hardware it's read-only)
  if ([4000006h] AND 1FFh)=10Eh ;VCOUNT (on real hardware it's 000h..106h)
    [4FFF010h]=13579BDFh        ;\initialize/reset something
    [4FFF010h]=FDB97531h        ;/
    Ensata=true
  else
    Ensata=false
  endif
```

Once when a commercial game has detected Ensata, it stops communicating with the ARM7, and instead it does seem to want to communicate with the Ensata executable (which has little to do with real NDS hardware). Ie. aside from "unlocking" port 4FFF000h, it does also "lock" access to the ARM7 hardware (like sound, touchscreen, RTC, etc).

ISD (Intelligent Systems Debugger or so) I/O Ports

The ISD ports seem to be real (non-emulated) debugging ports, mapped to the GBA Slot region at 8000000h-9FFFFFFh, and used to output text messages, and possible also other debugging stuff.
There are appear to be two variants: nitroemu and cgbemu (the latter appears to be dating back to old 8bit CGB hardware; which was apparently still used for the NDS two hardware generations later).

NDS Devkit

In Nintendo's devkit, debug messages are handled in file "os_printf.c", this file detects the available hardware/software based debug I/O ports, and redirects the [OS_PutString] vector to the corresponding string_out function (eg. to OS_PutStringAris for writing a 00h-terminated string to port 4FFF000h). With some minimal efforts, this could be redirected to the corresponding no$gba debug I/O ports.

DS Cartridges, Encryption, Firmware

Cartridges

DS Cartridge Header
DS Cartridge Secure Area
DS Cartridge Icon/Title
DS Cartridge Protocol
DS Cartridge Backup
DS Cartridge I/O Ports
DS Cartridge NitroROM and NitroARC File Systems
DS Cartridge PassMe/PassThrough
DS Cartridge GBA Slot

Add-Ons

DS Cart Rumble Pak
DS Cart Slider with Rumble
DS Cart Expansion RAM
DS Cart Unknown Extras

Special Cartridges

DS Cart Cheat Action Replay DS
DS Cart Cheat Codebreaker DS
DS Cart DLDI Driver

Encryption

DS Encryption by Gamecode/Idcode (KEY1)
DS Encryption by Random Seed (KEY2)

Firmware / Wifi Flash
DS Firmware Serial Flash Memory
DS Firmware Header
DS Firmware Wifi Calibration Data
DS Firmware Wifi Internet Access Points
DS Firmware User Settings
DS Firmware Extended Settings

DS Cartridge Header

Header Overview (loaded from ROM Addr 0 to Main RAM 27FFE00h on Power-up)

```
Address Bytes Expl.
000h    12    Game Title  (Uppercase ASCII, padded with 00h)
00Ch    4     Gamecode    (Uppercase ASCII, NTR-<code>)        (0=homebrew)
010h    2     Makercode   (Uppercase ASCII, eg. "01"=Nintendo) (0=homebrew)
012h    1     Unitcode    (00h=NDS, 02h=NDS+DSi, 03h=DSi) (bit1=DSi)
013h    1     Encryption Seed Select (00..07h, usually 00h)
014h    1     Devicecapacity         (Chipsize = 128KB SHL nn) (eg. 7 = 16MB)
015h    7     Reserved    (zero filled)
01Ch    1     Reserved    (zero)                       (except, used on DSi)
01Dh    1     NDS Region  (00h=Normal, 80h=China, 40h=Korea) (other on DSi)
01Eh    1     ROM Version (usually 00h)
01Fh    1     Autostart (Bit2: Skip "Press Button" after Health and Safety)
              (Also skips bootmenu, even in Manual mode & even Start pressed)
020h    4     ARM9 rom_offset    (4000h and up, align 1000h)
024h    4     ARM9 entry_address (2000000h..23BFE00h)
028h    4     ARM9 ram_address   (2000000h..23BFE00h)
02Ch    4     ARM9 size          (max 3BFE00h) (3839.5KB)
030h    4     ARM7 rom_offset    (8000h and up)
034h    4     ARM7 entry_address (2000000h..23BFE00h, or 37F8000h..3807E00h)
038h    4     ARM7 ram_address   (2000000h..23BFE00h, or 37F8000h..3807E00h)
03Ch    4     ARM7 size          (max 3BFE00h, or FE00h) (3839.5KB, 63.5KB)
040h    4     File Name Table (FNT) offset
044h    4     File Name Table (FNT) size
048h    4     File Allocation Table (FAT) offset
04Ch    4     File Allocation Table (FAT) size
050h    4     File ARM9 overlay_offset
054h    4     File ARM9 overlay_size
058h    4     File ARM7 overlay_offset
05Ch    4     File ARM7 overlay_size
060h    4     Port 40001A4h setting for normal commands (usually 00586000h)
064h    4     Port 40001A4h setting for KEY1 commands   (usually 001808F8h)
068h    4     Icon/Title offset (0=None) (8000h and up)
06Ch    2     Secure Area Checksum, CRC-16 of [[020h]..00007FFFh]
06Eh    2     Secure Area Delay (in 131kHz units) (051Eh=10ms or 0D7Eh=26ms)
070h    4     ARM9 Auto Load List Hook RAM Address (?) ;\endaddr of auto-load
074h    4     ARM7 Auto Load List Hook RAM Address (?) ;/functions
078h    8     Secure Area Disable (by encrypted "NmMdOnly") (usually zero)
080h    4     Total Used ROM size (remaining/unused bytes usually FFh-padded)
084h    4     ROM Header Size (4000h)
088h    28h   Reserved (zero filled; except, [88h..93h] used on DSi)
0B0h    10h   Reserved (zero filled; or "DoNotZeroFillMem"=unlaunch fastboot)
0C0h    9Ch   Nintendo Logo (compressed bitmap, same as in GBA Headers)
15Ch    2     Nintendo Logo Checksum, CRC-16 of [0C0h-15Bh], fixed CF56h
15Eh    2     Header Checksum, CRC-16 of [000h-15Dh]
160h    4     Debug rom_offset   (0=none) (8000h and up)       ;only if debug
164h    4     Debug size         (0=none) (max 3BFE00h)        ;version with
168h    4     Debug ram_address  (0=none) (2400000h..27BFE00h) ;SIO and 8MB
16Ch    4     Reserved (zero filled) (transferred, and stored, but not used)
170h    90h   Reserved (zero filled) (transferred, but not stored in RAM)
```

DSi Cartridges are using an extended cartridge header,
DSi Cartridge Header
Some of that new/changed DSi header entries are important even in NDS mode:
- On DSi, ARM9/ARM7 areas are restricted to 2.75MB (instead 3.8MB on real NDS)
- New NDS titles must have RSA signatures (and old titles must be in whitelist)

For more info about CRC-16, see description of GetCRC16 BIOS function,
BIOS Misc Functions
For the Logo checksum, the BIOS verifies only [15Ch]=CF56h, it does NOT verify the actual data at [0C0h-15Bh] (nor it's checksum), however, the data is verified by the firmware.

Secure Area Delay
The Secure Area Delay at header[06Eh] is counted in 130.912kHz units (which can be clocked via one of the hardware timers with prescaler=F/256 and reload=(10000h-((X AND 3FFFh)+2)); for some weird reason, in case of Header checksum it's ANDed with 1FFFh instead of 3FFFh). Commonly used values are X=051Eh (10ms), and X=0D7Eh (26ms).
The delay is used for all Blowfish encrypted commands, the actual usage/purpose differs depending on bit31 of the ROM Chip ID:
When ChipID.Bit31=0 (commands are sent ONCE): The delay is issued BEFORE sending the command:
```
Delay,Cmd
```
Older/newer games are using delays of 10ms/26ms (although all known existing cartridges with Bit31=0 would actually work WITHOUT delays).
When ChipID.Bit31=1 (commands are repeated MULTIPLE times): The delay is issued AFTER sending the command for the FIRST time:
```
Cmd,Delay,Cmd                                   ;for 2x repeat
Cmd,Delay,Cmd,Cmd,Cmd,Cmd,Cmd,Cmd,Cmd,Cmd       ;for 9x repeat
```
Known games are using delays of 26ms (although all known existing cartridges (=Cooking Coach) with Bit31=1 would actually work with shorter delays of ca. 7ms (but, better use 8ms for safety)).

NDS Gamecodes
This is the same code as the NTR-UTTD (NDS) or TWL-UTTD (DSi) code which is printed on the package and sticker on (commercial) cartridges (excluding the leading "NTR-" or "TWL-" part).
```
U  Unique Code          (usually "A", "B", "C", or special meaning)
TT Short Title          (eg. "PM" for Pac Man)
D  Destination/Language (usually "J" or "E" or "P" or specific language)
```
The first character (U) is usually "A" or "B", in detail:
```
A NDS common games
B NDS common games
C NDS common games
D DSi-exclusive games
H DSiWare (system utilities and browser) (eg. HNGP=browser)
I NDS and DSi-enhanced games with built-in Infrared port
K DSiWare (dsiware games and flipnote) (eg. KGUV=flipnote)
N NDS nintendo channel demo's japan (NTR-NTRJ-JPN)
T NDS many games
U NDS utilities, educational games, or uncommon extra hardware?
V DSi-enhanced games
Y NDS many games
```
The second/third characters (TT) are:
```
Usually an abbreviation of the game title (eg. "PM" for "Pac Man") (unless
that gamecode was already used for another game, then TT is just random)
```
The fourth character (D) indicates Destination/Language:
```
A Asian    E English/USA  I Italian   M Swedish  Q Danish   U Australian
B N/A      F French       J Japanese  N Nor      R Russian  V EUR+AUS
C Chinese  G N/A          K Korean    O Int      S Spanish  W..Z Europe #3..5
D German   H Dutch        L USA #2    P Europe   T USA+AUS
```

DS Cartridge Secure Area

The Secure Area is located in ROM at 4000h..7FFFh, it can contain normal program code and data, however, it can be used only for ARM9 boot code, it cannot be used for ARM7 boot code, icon/title, filesystem, or other data.

Secure Area Size
The Secure Area exists if the ARM9 boot code ROM source address (src) is located within 4000h..7FFFh, if so, it will be loaded (by BIOS via KEY1 encrypted commands) in 4K portions, starting at src, aligned by 1000h, up to address 7FFFh. The secure area size if thus 8000h-src, regardless of the ARM9 boot code size entry in header.
Note: The BIOS silently skips any NDS9 bootcode at src<4000h.
Cartridges with src>=8000h do not have a secure area.

Secure Area ID
The first 8 bytes of the secure area are containing the Secure Area ID, the ID is required (verified by BIOS boot code), the ID value changes during boot process:

```
Value                Expl.
"encryObj"           raw ID before encryption (raw ROM-image)
(encrypted)          encrypted ID after encryption (encrypted ROM-image)
"encryObj"           raw ID after decryption (verified by BIOS boot code)
E7FFDEFFh,E7FFDEFFh  destroyed ID (overwritten by BIOS after verify)
```

If the decrypted ID does match, then the BIOS overwrites the first 8 bytes by E7FFDEFFh-values (ie. only the ID is destroyed). If the ID doesn't match, then the first 800h bytes (2K) are overwritten by E7FFDEFFh-values.

Secure Area First 2K Encryption/Content
The first 2K of the Secure Area (if it exists) are KEY1 encrypted. In official games, this 2K region contains data like so (in decrypted form):

```
000h..007h  Secure Area ID (see above)
008h..00Dh  Fixed (FFh,DEh,FFh,E7h,FFh,DEh)
00Eh..00Fh  CRC16 across following 7E0h bytes, ie. [010h..7FFh]
010h..7FDh  Unknown/random values, mixed with some THUMB SWI calls
7FEh..7FFh  Fixed (00h,00h)
```

Of which, only the ID in the first 8 bytes is verified. Neither BIOS nor (current) firmare versions are verifying the data at 008h..7FFh, so the 7F8h bytes may be also used for normal program code/data.

Avoiding Secure Area Encryption
WLAN files are reportedly same format as cartridges, but without Secure Area, so games with Secure Area cannot be booted via WLAN. No$gba can encrypt and decrypt Secure Areas only if the NDS BIOS-images are present. And, Nintendo's devkit doesn't seem to support Secure Area encryption of unreleased games.
So, unencrypted cartridges are more flexible in use. Ways to avoid encryption (which still work on real hardware) are:
1) Set NDS9 ROM offset to 4000h, and leave the first 800h bytes of the Secure Area 00h-filled, which can be (and will be) safely destroyed during loading; due to the missing "encryObj" ID; that method is used by Nintendo's devkit.
2) Set NDS9 ROM offset to 8000h or higher (cartridge has no Secure Area at all).
3) Set NDS9 ROM offset, RAM address, and size to zero, set NDS7 ROM offset to 200h, and point both NDS9 and NDS7 entrypoints to the loaded NDS7 region. That method avoids waste of unused memory at 200h..3FFFh, and it should be compatible with the NDS console, however, it is not comaptible with commercial cartridges - which do silently redirect address below 4000h to "addr=8000h+(addr AND 1FFh)". Still, it should work with inofficial flashcards, which do not do that redirection. No$gba emulates the redirection for regular official cartridges, but it disables redirection for homebrew carts if NDS7 rom offset<8000h, and NDS7 size>0.
[One possible problem: Newer "anti-passme" firmware versions reportedly check that the entrypoint isn't set to 80000C0h, that firmwares might also reject NDS9 entrypoints within the NDS7 bootcode region?]

DS Cartridge Icon/Title

The ROM offset of the Icon/Title is defined in CartHdr[68h]. The size was originally implied by the size of the original Icon/Title structure rounded to 200h-byte sector boundary (ie. A00h bytes for Version 1 or 2), however, later DSi carts are having a size entry at CartHdr[208h] (usually 23C0h).
If it is present (ie. if CartHdr[68h]=nonzero), then Icon/Title are displayed in the bootmenu.

```
0000h 2     Version (0001h, 0002h, 0003h, or 0103h)
0002h 2     CRC16 across entries 0020h..083Fh (all versions)
0004h 2     CRC16 across entries 0020h..093Fh (Version 0002h and up)
0006h 2     CRC16 across entries 0020h..0A3Fh (Version 0003h and up)
0008h 2     CRC16 across entries 1240h..23BFh (Version 0103h and up)
000Ah 16h   Reserved (zero-filled)
0020h 200h  Icon Bitmap  (32x32 pix) (4x4 tiles, 4bit depth) (4x8 bytes/tile)
0220h 20h   Icon Palette (16 colors, 16bit, range 0000h-7FFFh)
            (Color 0 is transparent, so the 1st palette entry is ignored)
0240h 100h  Title 0 Japanese  (128 characters, 16bit Unicode)
0340h 100h  Title 1 English   ("")
0440h 100h  Title 2 French    ("")
0540h 100h  Title 3 German    ("")
0640h 100h  Title 4 Italian   ("")
0740h 100h  Title 5 Spanish   ("")
0840h 100h  Title 6 Chinese   ("")                (Version 0002h and up)
0940h 100h  Title 7 Korean    ("")                (Version 0003h and up)
0A40h 800h  Zerofilled (probably reserved for Title 8..15)
```

Below for animated DSi icons only (Version 0103h and up):

```
1240h 1000h Icon Animation Bitmap 0..7 (200h bytes each, format as above)
2240h 100h  Icon Animation Palette 0..7 (20h bytes each, format as above)
2340h 80h   Icon Animation Sequence (16bit tokens)
```

Unused/padding bytes:

```
0840h 1C0h  Unused/padding (FFh-filled) in Version 0001h
0940h C0h   Unused/padding (FFh-filled) in Version 0002h
23C0h 40h   Unused/padding (FFh-filled) in Version 0103h
```

Versions

```
0001h = Original
0002h = With Chinese Title
0003h = With Chinese+Korean Titles
0103h = With Chinese+Korean Titles and animated DSi icon
```

Title Strings

Usually, for non-multilanguage games, the same (english) title is stored in all title entries. The title may consist of ASCII characters 0020h-007Fh, character 000Ah (linefeed), and should be terminated/padded by 0000h.
The whole text should not exceed the dimensions of the DS cart field in the bootmenu (the maximum number of characters differs due to proportional font).
The title is usually split into a primary title, optional sub-title, and manufacturer, each separated by 000Ah character(s). For example: "America", 000Ah, "The Axis of War", 000Ah, "Cynicware", 0000h.

Icon Animation Sequence (DSi)

The sequence is represented by 16bit tokens, in the following format:

```
15    Flip Vertically   (0=No, 1=Yes)
14    Flip Horizontally (0=No, 1=Yes)
13-11 Palette Index     (0..7)
10-8  Bitmap Index      (0..7)
7-0   Frame Duration    (01h..FFh) (in 60Hz units)
```

Value 0000h indicates the end of the sequence. If the first token is 0000h, then the non-animated default image is shown.
Uh, actually, a non-animated icon uses values 01h,00h,00h,01h, followed by 7Ch zerofilled bytes (ie. 0001h, 0100h, 3Eh x 0000h)?

FAT16:\title\000300tt\4gggggggg\data\banner.sav ;if carthdr[1BFh].bit2=1
Some DSi games are having a separate "banner.sav" file stored in the eMMC filesystem, enabled via carthdr[1BFh].bit2 (allowing to indicate the game progress by overriding the default icon). The banner files are 4000h bytes in size, the animation data is same as above, but without title strings and without non-animated icon.

```
0000h 2     Version (0103h)
0002h 6     Reserved (zero-filled)
0008h 2     CRC16 across entries 0020h..119Fh (with initial value FFFFh)
000Ah 16h   Reserved (zero-filled)
0020h 1000h Icon Animation Bitmap 0..7 (200h bytes each)  ;\same format as
1020h 100h  Icon Animation Palette 0..7 (20h bytes each)  ; in Icon/Title
1120h 80h   Icon Animation Sequence (16bit tokens)        ;/
11A0h 2E60h Garbage (random values, maybe due to eMMC decryption)
```

The feature is used by some Brain Age Express games (for example, Brain Age Express Sudoku: 'title\00030004\4b4e3945\data\banner.sav').
The feature does probably work only for DSiware titles (unless there are any DSi carts with SD/MMC access enabled; or unless there is a feature for storing similar data in cartridge memory).

DS Cartridge Protocol

Communication with Cartridge ROM relies on sending 8 byte commands to the cartridge, after the sending the command, a data stream can be received from the cartridge (the length of the data stream isn't fixed, below descriptions show the default length in brackets, but one may receive more, or less bytes, if desired).

Cartridge Memory Map

```
0000000h-0000FFFh Header (unencrypted)
0001000h-0003FFFh Not read-able (zero filled in ROM-images)
0004000h-0007FFFh Secure Area, 16KBytes (first 2Kbytes with extra encryption)
0008000h-...      Main Data Area
```

DSi cartridges are split into a NDS area (as above), and a new DSi area:

```
XX00000h XX02FFFh DSi Not read-able (XX00000h=first megabyte after NDS area)
XX03000h-XX06FFFh DSi ARM9i Secure Area (usually with modcrypt encryption)
XX07000h-...      DSi Main Data Area
```

Cartridge memory must be copied to RAM (the CPU cannot execute code in ROM).

Command Summary, Cmd/Reply-Encryption Type, Default Length

```
Command/Params    Expl.                              Cmd  Reply Len
-- Unencrypted Load --
9F00000000000000h Dummy (read HIGH-Z bytes)          RAW  RAW   2000h
0000000000000000h Get Cartridge Header               RAW  RAW   200h DSi:1000h
9000000000000000h 1st Get ROM Chip ID                RAW  RAW   4
A000000000000000h Get 3DS encryption type (3DS)      RAW  RAW   4
00aaaaaaaa000000h Unencrypted Data (debug ver only) RAW  RAW   200h
3Ciiijjjxkkkkkxxh Activate KEY1 Encryption (NDS)     RAW  RAW   0
3Diiijjjxkkkkkxxh Activate KEY1 Encryption (DSi)     RAW  RAW   0
3E00000000000000h Activate 16-byte commands (3DS)    RAW  RAW   0
-- Secure Area Load --
4llllmmmnnnkkkkkh Activate KEY2 Encryption Mode      KEY1 FIX   910h+0
1lllliiijjjkkkkkh 2nd Get ROM Chip ID                KEY1 KEY2  910h+4
xxxxxxxxxxxxxxxxh Invalid - Get KEY2 Stream XOR 00h KEY1 KEY2  910h+...
2bbbbiiijjjkkkkkh Get Secure Area Block (4Kbytes)    KEY1 KEY2  910h+10A8h
6lllliiijjjkkkkkh Optional KEY2 Disable              KEY1 KEY2  910h+?
Allliiijjjkkkkkh Enter Main Data Mode                KEY1 KEY2  910h+0
-- Main Data Load --
B7aaaaaaaa000000h Encrypted Data Read                KEY2 KEY2  200h
B8000000000000000h 3rd Get ROM Chip ID               KEY2 KEY2  4
xxxxxxxxxxxxxxxxh Invalid - Get KEY2 Stream XOR 00h KEY2 KEY2  ...
B5000000000000000h Whatever NAND related? (DSi?)     KEY2 KEY2  0
D6000000000000000h Whatever NAND related? (DSi?)     KEY2 KEY2  4
```

The parameter digits contained in above commands are:

```
aaaaaaaa     32bit ROM address (command B7 can access only 8000h and up)
bbbb         Secure Area Block number (0004h..0007h for addr 4000h..7000h)
x,xx         Random, not used in further commands (DSi: always zero)
iii,jjj,llll Random, must be SAME value in further commands
kkkkk        Random, must be INCREMENTED after FURTHER commands
mmm,nnn      Random, used as KEY2-encryption seed
```

++++ Unencrypted Commands (First Part of Boot Procedure) ++++

Cartridge Reset

The /RES Pin switches the cartridge into unencrypted mode. After reset, the first two commands (9Fh and 00h) are transferred at 4MB/s CLK rate.

9F00000000000000h (2000h) - Dummy

Dummy command send after reset, returns endless stream of HIGH-Z bytes (ie. usually receiving FFh, immediately after sending the command, the first 1-2 received bytes may be equal to the last command byte).

0000000000000000h (200h) (DSi:1000h) - Get Header

Returns RAW unencrypted cartridge header, repeated every 1000h bytes. The interesting area are the 1st 200h bytes, the rest is typically zero filled (except on DSi carts, which do use the whole 1000h bytes).
The Gamecode header entry is used later on to initialize the encryption. Also, the ROM Control entries define the length of the KEY1 dummy periods (typically 910h clocks), and the CLK transfer rate for further commands (typically faster than the initial 4MB/s after power up).

9000000000000000h (4) - 1st Get ROM Chip ID

Returns RAW unencrypted Chip ID (eg. C2h,0Fh,00h,00h), repeated every 4 bytes.

```
1st byte - Manufacturer (eg. C2h=Macronix) (roughly based on JEDEC IDs)
2nd byte - Chip size (00h..7Fh: (N+1)Mbytes, F0h..FFh: (100h-N)*256Mbytes?)
3rd byte - Flags (see below)
4th byte - Flags (see below)
```

The Flag Bits in 3th byte can be

```
0   Maybe Infrared flag? (in case ROM does contain on-chip infrared stuff)
1   Unknown (set in some 3DS carts)
2-7 Zero
```

The Flag Bits in 4th byte can be

```
0-2 Zero
3   Seems to be NAND flag (0=ROM, 1=NAND) (observed in only ONE cartridge)
4   3DS Flag (0=NDS/DSi, 1=3DS)
5   Zero   ... set in ... DSi-exclusive games? (support cmd B5h/D6h ?)
6   DSi flag (0=NDS/3DS, 1=DSi)
7   Cart Protocol Variant (0=older/smaller carts, 1=newer/bigger carts)
```

Existing/known ROM IDs are:

```
C2h,07h,00h,00h NDS Macronix 8MB ROM  (eg. DS Vision)
AEh,0Fh,00h,00h NDS Noname   16MB ROM (eg. Meine Tierarztpraxis)
C2h,0Fh,00h,00h NDS Macronix 16MB ROM (eg. Metroid Demo)
C2h,1Fh,00h,00h NDS Macronix 32MB ROM (eg. Over the Hedge)
C2h,1Fh,00h,40h DSi Macronix 32MB ROM (eg. Art Academy, TWL-VAAV, SystemFlaw)
80h,3Fh,01h,E0h ?            64MB ROM+Infrared (eg. Walk with Me, NTR-IMWP)
AEh,3Fh,00h,E0h DSi Noname   64MB ROM (eg. de Blob 2, TWL-VD2V)
C2h,3Fh,00h,00h NDS Macronix 64MB ROM (eg. Ultimate Spiderman)
C2h,3Fh,00h,40h DSi Macronix 64MB ROM (eg. Crime Lab, NTR-VAOP)
80h,7Fh,00h,80h NDS SanDisk  128MB ROM (DS Zelda, NTR-AZEP-0)
80h,7Fh,01h,E0h ?            128MB ROM+Infrared? (P-letter Soul Silver, IPGE)
C2h,7Fh,00h,80h NDS Macronix 128MB ROM (eg. Spirit Tracks, NTR-BKIP)
C2h,7Fh,00h,C0h DSi Macronix 128MB ROM (eg. Cooking Coach/TWL-VCKE)
ECh,7Fh,00h,88h NDS Samsung  128MB NAND (eg. Warioware D.I.Y.)
ECh,7Fh,01h,88h NDS Samsung? 128MB NAND+What? (eg. Jam with the Band, UXBP)
ECh,7Fh,00h,E8h DSi Samsung? 128MB NAND (eg. Face Training, USKV)
80h,FFh,80h,E0h NDS          256MB ROM (Kingdom Hearts - Re-Coded, NTR-BK9P)
```

```
C2h,FFh,01h,C0h DSi Macronix 256MB ROM+Infrared? (eg. P-Letter White)
C2h,FFh,00h,80h NDS Macronix 256MB ROM (eg. Band Hero, NTR-BGHP)
C2h,FEh,01h,C0h DSi Macronix 512MB ROM+Infrared? (eg. P-Letter White 2)
C2h,FEh,00h,90h 3DS Macronix probably 512MB? ROM (eg. Sims 3)
45h,FAh,00h,90h 3DS SunDisk? maybe... 1.5GB? ROM (eg. Starfox)
C2h,F8h,00h,90h 3DS Macronix maybe... 2GB?   ROM (eg. Kid Icarus)
C2h,7Fh,00h,90h 3DS Macronix 128MB ROM CTR-P-AENJ MMinna no Ennichi
C2h,FFh,00h,90h 3DS Macronix 256MB ROM CTR-P-AFSJ Pro Yakyuu Famista 2011
C2h,FEh,00h,90h 3DS Macronix 512MB ROM CTR-P-AFAJ Real 3D Bass FishingFishOn
C2h,FAh,00h,90h 3DS Macronix 1GB ROM CTR-P-ASUJ Hana to Ikimono Rittai Zukan
C2h,FAh,02h,90h 3DS Macronix 1GB ROM CTR-P-AGGW Luigis Mansion 2 ASiA CHT
C2h,F8h,00h,90h 3DS Macronix 2GB ROM CTR-P-ACFJ Castlevania - Lords of Shadow
C2h,F8h,02h,90h 3DS Macronix 2GB ROM CTR-P-AH4J Monster Hunter 4
AEh,FAh,00h,90h 3DS          1GB ROM CTR-P-AGKJ Gyakuten Saiban 5
AEh,FAh,00h,98h 3DS          1GB NAND CTR-P-EGDJ Tobidase Doubutsu no Mori
45h,FAh,00h,90h 3DS          1GB ROM CTR-P-AFLJ Fantasy Life
45h,F8h,00h,90h 3DS          2GB ROM CTR-P-AVHJ Senran Kagura Burst - Guren
C2h,F0h,00h,90h 3DS Macronix 4GB ROM CTR-P-ABRJ Biohazard Revelations
FFh,FFh,FFh,FFh None (no cartridge inserted)
```

The Samsung NAND chip appears to use a slightly different protocol (seems as if it allows to read ROM header and ID only once, or as if it gets confused when reading more than 4 ID bytes, or so) (and of course, the protocol is somehow extended, allowing to write data to the NAND memory). The official JEDEC ID for Samsung would be "CEh", but for some reason, Samsung's NDS chip does spit out "ECh" as Maker ID.
ID "45h" ("SunDisk" according to a JEDEC ID list) might refer to "SanDisk"?

3Ciiijjjxkkkkkxxh (0) - Activate KEY1 Encryption Mode
The 3Ch command returns endless stream of HIGH-Z bytes, all following commands, and their return values, are encrypted. The random parameters iii,jjj,kkkkk must be re-used in further commands; the 20bit kkkkk value is to be incremented by one after each <further> command (it is <not> incremented after the 3Ch command).

3Diiijjjxkkkkkxxh (0) - Activate KEY1 Encryption Mode and Unlock DSi Mode
Same as command 3Ch (but with different initial 1048h-byte encryption values), and works only on DSi carts.
Command 3Dh is unlocking two features on DSi carts:

```
1) Command 2bbbbiiijjjkkkkkh loads ARM9i secure area (instead of ARM9 area)
2) Command B7aaaaaaaa000000h allows to read the 'whole' cartridge space
```

Without command 3Dh, DSi carts will allow to read only the first some megabytes (for example, the first 11 Mbyte of the System Flaw cartridge), and the remaining memory returns mirrors of "addr=8000h+(addr AND 1FFh)").
Note: After reset, the cartridge protocol allows to send only either one of the 3Ch/3Dh commands (DSi consoles can control the cartridge reset pin, so they can first send 3Ch and read the normal secure area, then issue a reset and 3Dh and read the DSi secure area) (on a NDS one could do the same by ejecting/inserting the cartridge instead of toggling the reset pin).

++++ KEY1 Encrypted Commands (2nd Part of Boot procedure) ++++

4llllmmmnnnkkkkkh (910h) - Activate KEY2 Encryption Mode
KEY1 encrypted command, parameter mmmnnn is used to initialize the KEY2 encryption stream. Returns 910h dummy bytes (which are still subject to old KEY2 settings; at pre-initialization time, this is fixed: HIGH-Z, C5h, 3Ah, 81h, etc.). The new KEY2 seeds are then applied, and the first KEY2 byte is then precomputed. The 910h dummy stream is followed by that precomputed byte value endless repeated (this is the same value as that "underneath" of the first HIGH-Z dummy-byte of the next command).
Secure1000h: Returns repeated FFh bytes (instead of the leading C5h, 3Ah, 81h, etc. stuff).
Secure1000h: Returns repeated FFh bytes (instead of the repeated precomputed value).

1llliiijjjkkkkkh (914h) - 2nd Get ROM Chip ID / Get KEY2 Stream
KEY1 encrypted command. Returns 910h dummy bytes, followed by KEY2 encrypted Chip ID repeated every 4 bytes, which must be identical as for the 1st Get ID command. The BIOS randomly executes this command once

or twice. Changing the first command byte to any other value returns an endless KEY2 encrypted stream of 00h bytes, that is the easiest way to retrieve encryption values and to bypass the copyprotection.

2bbbbiiijjjkkkkkh (19B8h) - Get Secure Area Block
KEY1 encrypted command. Used to read a secure area block (bbbb in range 0004h..0007h for addr 4000h..7000h) (or, after sending command 3Dh on a DSi: bbbb in range 0004h..0007h for addr XX03000h..XX06000h).
Each block is 4K, so it requires four Get Secure Area commands to receive the whole Secure Area (ROM locations 4000h-7FFFh), the BIOS is reading these blocks in random order.
Normally (if the upper bit of the Chip ID is set): Returns 910h dummy bytes, followed by 200h KEY2 encrypted Secure Area bytes, followed by 18h KEY2 encrypted 00h bytes, then the next 200h KEY2 encrypted Secure Area bytes, again followed by 18h KEY2 encrypted 00h bytes, and so on. That stream is repeated every 10C0h bytes (8x200h data bytes, plus 8x18h zero bytes).
Alternately (if the upper bit of the Chip ID is zero): Returns 910h dummy bytes, followed by 1000h KEY2 encrypted Secure Area bytes, presumably followed by 18h bytes, too.
Aside from above KEY2 encryption (which is done by hardware), the first 2K of the NDS Secure Area is additionally KEY1 encrypted; which must be resolved after transfer by software (and the DSi Secure Area is usually modcrypted, as specified in the cartridge header).

6lllliiijjjkkkkkh (0) - Optional KEY2 Disable
KEY1 encrypted command. Returns 910h dummy bytes (which are still KEY2 affected), followed by endless stream of RAW 00h bytes. KEY2 encryption is disabled for all following commands.
This command is send only if firmware[18h] matches encrypted string "enPngOFF", and ONLY if firmware get_crypt_keys had completed BEFORE completion of secure area loading, this timing issue may cause unstable results.

Allllliiijjjkkkkkh (910h) - Enter Main Data Mode
KEY1 encrypted command. Returns 910h dummy bytes, followed by endless KEY2 encrypted stream of 00h bytes. All following commands are KEY2 encrypted.

++++ KEY2 Encrypted Commands (Main Data Transfer) ++++

B7aaaaaaaa000000h (200h) - Get Data
KEY2 encrypted command. The desired ROM address is specifed, MSB first, in parameter bytes (a). Returned data is KEY2 encrypted.
There is no alignment restriction for the address. However, the datastream wraps to the begin of the current 4K block when address+length crosses a 4K boundary (1000h bytes).
The command can be used only for addresses 8000h and up. Addresses 0..7FFFh are silently redirected to address "8000h+(addr AND 1FFh)". DSi cartridges will also reject XX00000h..XX06FFFh in the same fashion (and also XX07000h and up if the DSi cartridge isn't unlocked via command 3Dh).
Addresses that do exceed the ROM size do mirror to the valid address range (that includes mirroring non-loadable regions like 0..7FFFh to "8000h+(addr AND 1FFh)"; some newer games are using this behaviour for some kind of anti-piracy checks).

B800000000000000h (4) - 3rd Get ROM Chip ID
KEY2 encrypted command. Returns KEY2 encrypted Chip ID repeated every 4 bytes.

xxxxxxxxxxxxxxxxh - Invalid Command
Any other command (anything else than above B7h and B8h) in KEY2 command mode causes communcation failures. The invalid command returns an endless KEY2 encrypted stream of 00h bytes. After the invalid command, the KEY2 stream is NOT advanced for further command bytes, further commands seems to return KEY2 encrypted 00h bytes, of which, the first returned byte appears to be HIGH-Z.
Ie. the cartridge seems to have switched back to a state similar to the KEY1-phase, although it doesn't seem to be possible to send KEY1 commands.

++++ Notes ++++

KEY1 Command Encryption / 910h Dummy Bytes
All KEY1 encrypted commands are followed by 910h dummy byte transfers, these 910h clock cycles are probably used to decrypt the command at the cartridge side; communication will fail when transferring less than 910h bytes.
The return values for the dummy transfer are: A single HIGH-Z byte, followed by 90Fh KEY2-encrypted 00h bytes. The KEY2 encryption stream is advanced for all 910h bytes, including for the HIGH-Z byte.
Note: Current cartridges are using 910h bytes, however, other carts might use other amounts of dummy bytes, the 910h value can be calculated based on ROM Control entries in cartridge header. For the KEY1 formulas, see:
DS Encryption by Gamecode/Idcode (KEY1)

KEY2 Command/Data Encryption
DS Encryption by Random Seed (KEY2)

Cart Protocol Variants (Chip ID.Bit31)
There are two protocol variants for NDS carts, indicated by Bit31 of the ROM Chip ID (aka bit7 of the 4th ID byte):

```
1) Chip ID.Bit31=0  Used by older/smaller carts with up to 64MB ROM
2) Chip ID.Bit31=1  Used by newer/bigger carts with 64MB or more ROM
```

The first variant (for older carts) is described above. The second second variant includes some differences for KEY1 encrypted commands:
GAPS: The commands have the same 910h-cycle gaps, but without outputting CLK pulses during those gaps (ie. used with ROMCTRL.Bit28=0) (the absence of the CLKs implies that there is no dummy data transferred during gaps, and accordingly, that the KEY2 stream isn't advanced during the 910h gap cycles).
REPEATED COMMANDS and SECURE AREA DELAY: All KEY1 encrypted commands must be sent TWICE (or even NINE times). First, send the command with 0-byte Data transfer length. Second, issue the Secure Area Delay (required; use the delay specified in cart header[06Eh]).
Third, send the command once again with 0-byte or 4-byte data transfer length (usually 0 bytes, or 4-bytes for Chip ID command), or sent it eight times with 200h-byte data transfer length (for the 1000h-byte secure area load command).
For those repeats, always resend exactly the same command (namely, kkkkk is NOT incremented during repeats, and there is no extra index needed to select 200h-byte portions within 1000h-byte blocks; the cartridge is automatically outputting the eight portions one after another).

DS Cartridge Backup

SPI Bus Backup Memory

Type	Total Size	Page Size	Chip/Example	Game/Example
EEPROM	0.5K bytes	16 bytes	ST M95040-W	(eg. Metroid Demo)
EEPROM	8K bytes	32 bytes	ST M95640-W	(eg. Super Mario DS)
EEPROM	64K bytes	128 bytes	ST M95512-W	(eg. Downhill Jam)
FLASH	256K bytes	256 bytes	ST M45PE20	(eg. Skateland)
FLASH	256K bytes		Sanyo LE25FW203T	(eg. Mariokart)
FLASH	512K bytes	256 bytes	ST M25PE40?	(eg. which/any games?)
FLASH	512K bytes		ST 45PE40V6	(eg. DS Zelda, NTR-AZEP-0)
FLASH	1024K bytes		ST 45PE80V6	(eg. Spirit Tracks, NTR-BKIP)
FLASH	8192K bytes		MX25L6445EZNI-10G	(Art Academy only, TWL-VAAV)
FRAM	8K bytes	No limit	?	(eg. which/any games?)
FRAM	32K bytes	No limit	Ramtron FM25L256?	(eg. which/any games?)

Lifetime Stats

Type	Max Writes per Page	Data Retention

```
EEPROM    100,000               40 years
FLASH     100,000               20 years
FRAM      No limit              10 years
```

SPI Bus Backup Memory is accessed via Ports 40001A0h and 40001A2h, see
DS Cartridge I/O Ports

Commands

For all EEPROM and FRAM types:

```
06h WREN  Write Enable               Cmd, no parameters
04h WRDI  Write Disable              Cmd, no parameters
05h RDSR  Read Status Register       Cmd, read repeated status value(s)
01h WRSR  Write Status Register      Cmd, write one-byte value
9Fh RDID  Read JEDEC ID (not supported on EEPROM/FLASH, returns FFh-bytes)
```

For 0.5K EEPROM (8+1bit Address):

```
03h RDLO  Read from Memory 000h-0FFh  Cmd, addr lsb, read byte(s)
0Bh RDHI  Read from Memory 100h-1FFh  Cmd, addr lsb, read byte(s)
02h WRLO  Write to Memory 000h-0FFh   Cmd, addr lsb, write 1..MAX byte(s)
0Ah WRHI  Write to Memory 100h-1FFh   Cmd, addr lsb, write 1..MAX byte(s)
```

For 8K..64K EEPROM and for FRAM (16bit Address):

```
03h RD    Read from Memory            Cmd, addr msb,lsb, read byte(s)
02h WR    Write to Memory             Cmd, addr msb,lsb, write 1..MAX byte(s)
```

Note: MAX = Page Size (see above chip list) (no limit for FRAM).

For FLASH backup, commands should be same as for Firmware FLASH memory:
DS Firmware Serial Flash Memory

Status Register

```
0   WIP  Write in Progress (1=Busy) (Read only) (always 0 for FRAM chips)
1   WEL  Write Enable Latch (1=Enable) (Read only, except by WREN,WRDI)
2-3 WP   Write Protect (0=None, 1=Upper quarter, 2=Upper Half, 3=All memory)
```

For 0.5K EEPROM:

```
4-7 ONEs Not used (all four bits are always set to "1" each)
```

For 8K..64K EEPROM and for FRAM:

```
4-6 ZERO Not used (all three bits are always set to "0" each)
7   SRWD Status Register Write Disable (0=Normal, 1=Lock) (Only if /W=LOW)
```

WEL gets reset on Power-up, WRDI, WRSR, WRITE/LO/HI, and on /W=LOW.
The WRSR command allows to change ONLY the two WP bits, and the SRWD bit (if any), these bits are non-volatile (remain intact during power-down), respectively, the WIP bit must be checked to sense WRSR completion.

Detection (by examining hardware responses)

The overall memory type and bus-width can be detected by RDSR/RDID commands:

```
RDSR  RDID          Type         (bus-width)
FFh,  FFh,FFh,FFh   None         (none)
F0h,  FFh,FFh,FFh   EEPROM       (with 8+1bit address bus)
00h,  FFh,FFh,FFh   EEPROM/FRAM  (with 16bit address bus)
00h,  xxh,xxh,xxh   FLASH        (usually with 24bit address bus)
```

And, the RD commands can be used to detect the memory size/mirrors (though that won't work if the memory is empty).

Pin-Outs for EEPROM and FRAM chips

```
Pin Name Expl.
1  /S    Chip Select
2  Q     Data Out
3  /W    Write-Protect (not used in NDS, wired to VCC)
4  VSS   Ground
5  D     Data In
6  C     Clock
```

```
  7  /HOLD Transfer-pause (not used in NDS, wired to VCC)
  8  VCC   Supply 2.5 to 5.5V for M95xx0-W
```

FRAM (Ferroelectric Nonvolatile RAM) is fully backwards compatible with normal EEPROMs, but comes up with faster write/erase time (no delays), and with lower power consumption, and unlimited number of write/erase cycles. Unlike as for normal RAM, as far as I understand, the data remains intact without needing any battery.

Other special save memory

```
  DS Vision (NDS cart with microSD slot... and maybe ALSO with EEPROM?)
  Warioware D.I.Y. (uses a single NAND FLASH chip for both 'ROM' and 'SAVE')
    (the warioware chip is marked "SAMSUNG 004, KLC2811ANB-P204, NTR-UORE-0")
    (the warioware PCB is marked "DI X-7 C17-01")
  and, a few games are said to have "Flash - 64 Mbit" save memory?
```

DSi Internal eMMC and External SD Card

DSi cartridges are usually (maybe always) having SD/MMC access disabled, so they must stick using EEPROM/FLASH chips inside of the cartridges (which is required for NDS compatibility anyways). However, DSiware games (downloaded from DSi Shop) are allowed to save data on eMMC, using "private.sav" or "public.sav" files in their data folder. The size of that files is preset in cartridge header.

DS Cartridge I/O Ports

The Gamecard bus registers can be mapped to NDS7 or NDS9 via EXMEMCNT, see
DS Memory Control

40001A0h - NDS7/NDS9 - AUXSPICNT - Gamecard ROM and SPI Control (R/W)

```
  0-1   SPI Baudrate        (0=4MHz/Default, 1=2MHz, 2=1MHz, 3=512KHz)
  2-5   Not used            (always zero)
  6     SPI Hold Chipselect (0=Deselect after transfer, 1=Keep selected)
  7     SPI Busy            (0=Ready, 1=Busy) (presumably Read-only)
  8-12  Not used            (always zero)
  13    NDS Slot Mode       (0=Parallel/ROM, 1=Serial/SPI-Backup)
  14    Transfer Ready IRQ  (0=Disable, 1=Enable) (for ROM, not for AUXSPI)
  15    NDS Slot Enable     (0=Disable, 1=Enable) (for both ROM and AUXSPI)
```

The "Hold" flag should be cleared BEFORE transferring the LAST data unit, the chipselect will be then automatically cleared after the transfer, the program should issue a WaitByLoop(12) on NDS7 (or longer on NDS9) manually AFTER the LAST transfer.

40001A2h - NDS7/NDS9 - AUXSPIDATA - Gamecard SPI Bus Data/Strobe (R/W)

The SPI transfer is started on writing to this register, so one must <write> a dummy value (should be zero) even when intending to <read> from SPI bus.

```
  0-7  Data
  8-15 Not used (always zero)
```

During transfer, the Busy flag in AUXSPICNT is set, and the written DATA value is transferred to the device (via output line), simultaneously data is received (via input line). Upon transfer completion, the Busy flag goes off, and the received value can be then read from AUXSPIDATA, if desired.

40001A4h - NDS7/NDS9 - ROMCTRL - Gamecard Bus ROMCTRL (R/W)

```
  Bit   Expl.
  0-12  KEY1 gap1 length  (0-1FFFh) (forced min 08F8h by BIOS) (leading gap)
  13    KEY2 encrypt data (0=Disable, 1=Enable KEY2 Encryption for Data)
  14     "SE" Unknown? (usually same as Bit13) (does NOT affect timing?)
  15    KEY2 Apply Seed   (0=No change, 1=Apply Encryption Seed) (Write only)
  16-21 KEY1 gap2 length  (0-3Fh)   (forced min 18h by BIOS) (200h-byte gap)
  22    KEY2 encrypt cmd  (0=Disable, 1=Enable KEY2 Encryption for Commands)
  23    Data-Word Status  (0=Busy, 1=Ready/DRQ) (Read-only)
```

```
  24-26 Data Block size   (0=None, 1..6=100h SHL (1..6) bytes, 7=4 bytes)
  27    Transfer CLK rate (0=6.7MHz=33.51MHz/5, 1=4.2MHz=33.51MHz/8)
  28    KEY1 Gap CLKs (0=Hold CLK High during gaps, 1=Output Dummy CLK Pulses)
  29    RESB Release Reset (0=Reset, 1=Release) (cannot be cleared once set)
  30     "WR"   Unknown, maybe data-write? (usually 0) (read/write-able)
  31    Block Start/Status (0=Ready, 1=Start/Busy) (IRQ See 40001A0h/Bit14)
```

The cartridge header is booted at 4.2MHz CLK rate, and following transfers are then using ROMCTRL settings specified in cartridge header entries [060h] and [064h], which are usually using 6.7MHz CLK rate for the main data transfer phase (whereof, older MROM carts can actually transfer 6.7Mbyte/s, but newer 1T-ROM carts default to reading 200h-byte blocks with gap1=657h, thus reaching only 1.6Mbyte/s).

Transfer length of null, four, and 200h..4000h bytes are supported by the console, however, retail cartridges cannot cross 1000h-byte boundaries.

Default cart header entries

```
  hdr[60h]   hdr[64h]   hdr[6Eh]
  00586000h  001808F8h  051Eh     ;older/faster MROM
  00416657h  081808F8h  0D7Eh     ;newer/slower 1T-ROM
  ?          ?          ?         ;whatever NAND
```

Older/Faster MROM

The romctrl values in cartheader[60h,64h] are okay, but the secure delay in [6Eh] is nonsense (should be zero).

Misdeclared MROM

Some carts like SystemFlaw and BiggestLoser are actually containing MROM chips despite of having 1T-ROM values in cart header (gap1=657h is making loading insane slow, gap2=01h causes errors on 1000h-byte blocks, and secure.clk=4.2MHz is slowing down secure area loading, combined with even slower secure area delay despite of not needing any delay for MROM).

As the cart header entries are wrong, some other detection is needed: This can be probably done by checking ChipID.bit31 (or otherwise by testing if 1000h-block reading works with gap1=01h, if so, then it's 1T-ROM).

Newer/Slower 1T-ROM

Actual 1T-ROM carts can be very slow, especially when using the insane cart header values and default firmware blocksize of 200h bytes which drops loading speed from 6.7Mbytes/s to 1.6Mbyte/s (as workaround, use gap1=180h, blocksize=1000h, also secure area delay should be 400h, not D7Eh) (tested/working for CookingCoach, unknown if that timings work for all other carts).

NAND

Some cartridges are said to contain NAND memory, unknown if that's accessed with the normal ROM reading commands, and if so, with which timings.

Cart Reset

Reset flag in bit29 can be set once only (to release reset), the only way to clear the bit is power-off. However, there are some ways to issue resets:

1) On NDS: Manually eject/insert the cart (that won't affect bit29, but the cart will reset itself anyways upon power loss) (eject on DSi will power-off the cart slot).

2) If one of the two ROMCTRL registers (on ARM7 and ARM9) is still zero: Temporarily toggle ARM7/ARM9 cart access via EXMEMCNT on ARM9 side.

3) On DSi: If the 2nd cart slot ROMCTRL register (40021A4h) is still zero: Temporarily swap 1ns/2nd cart slot via SCFG_MC.bit15 on ARM7 side.

4) On DSi: Use SCFG_MC to toggle cart power off/on; this will actually reset bit29, the DSi firmware is actually using that method, but it's very slow (takes about 300ms, for the power-off wait, plus (unneccassary) hardcoded power-on delays).

40001A8h - NDS7/NDS9 - Gamecard bus 8-byte Command Out

The separate commands are described in the Cartridge Protocol chapter, however, once when the BIOS boot procedure has completed, one would usually only need command "B7aaaaaaaa000000h", for reading data (usually 200h bytes) from address aaaaaaaah (which should be usually aligned by 200h).

```
  0-7   1st Command Byte (at 40001A8h) (eg. B7h) (MSB)
  8-15  2nd Command Byte (at 40001A9h) (eg. addr bit 24-31)
  16-23 3rd Command Byte (at 40001AAh) (eg. addr bit 16-23)
  24-31 4th Command Byte (at 40001ABh) (eg. addr bit 8-15) (when aligned=even)
  32-39 5th Command Byte (at 40001ACh) (eg. addr bit 0-7)  (when aligned=00h)
  40-47 6th Command Byte (at 40001ADh) (eg. 00h)
```

```
  48-57 7th Command Byte (at 40001AEh) (eg. 00h)
  56-63 8th Command Byte (at 40001AFh) (eg. 00h) (LSB)
```
Observe that the command/parameter MSB is located at the smallest memory location (40001A8h), ie. compared with the CPU, the byte-order is reversed.

4100010h - NDS7/NDS9 - Gamecard bus 4-byte Data In (R)
```
  0-7   1st received Data Byte (at 4100010h)
  8-15  2nd received Data Byte (at 4100011h)
  16-23 3rd received Data Byte (at 4100012h)
  24-31 4th received Data Byte (at 4100013h)
```
After sending a command, data can be read from this register manually (when the DRQ bit is set), or by DMA (with DMASAD=4100010h, Fixed Source Address, Length=1, Size=32bit, Repeat=On, Mode=DS Gamecard).

40001B0h - 32bit - NDS7/NDS9 - Encryption Seed 0 Lower 32bit (W)
40001B4h - 32bit - NDS7/NDS9 - Encryption Seed 1 Lower 32bit (W)
40001B8h - 16bit - NDS7/NDS9 - Encryption Seed 0 Upper 7bit (bit7-15 unused)
40001BAh - 16bit - NDS7/NDS9 - Encryption Seed 1 Upper 7bit (bit7-15 unused)
These registers are used by the NDS7 BIOS to initialize KEY2 encryption (and there's normally no need to change that initial settings). Writes to the Seed registers do not have direct effect on the internal encryption registers, until the Seed gets applied by writing "1" to ROMCTRL.Bit15.
```
 For more info:
```
DS Encryption by Random Seed (KEY2)
Note: There are <separate> Seed registers for both NDS7 and NDS9, which can be applied by ROMCTRL on NDS7 and NDS9 respectively (however, once when applied to the internal registers, the new internal setting is used for <both> CPUs).

DS Cartridge NitroROM and NitroARC File Systems

The DS hardware, BIOS, and Firmware do NOT contain any built-in filesystem functions. The ARM9/ARM7 boot code (together max 3903KB), and Icon/Title information are automatically loaded on power-up. Programs that require to load additional data from cartridge ROM may do that either by implementing whatever functions to translate filenames to ROM addresses, or by reading from ROM directly.

NitroROM
The NitroROM Filesystem is used by many NDS games (at least those that have been developed with Nintendo's tools). It's used for ROM Cartridges, and, on the DSi, it's also used for DSiWare games (in the latter case, NitroROM acts as a 2nd virtual filesystem inside of the DSi's FAT16 filesystem).
```
  FNT = cart_hdr[040h]     ;\origin as defined in ROM cartridge header
  FAT = cart_hdr[048h]     ;/
  IMG = 00000000h          ;-origin at begin of ROM
```
Aside from using filenames, NitroROM files can be alternately accessed via Overlay IDs (see later on below).

NitroARC (Nitro Archive)
NARC Files are often found inside of NitroROM Filesystems (ie. NARC is a second virtual filesystem, nested inside of the actual filesystem). The NARC Format is very similar to the NitroROM Format, but with additional Chunk Headers (instead of the Cartridge ROM Header).
```
  ...  ...  Optional Header (eg. compression header, or RSA signature)
  000h 4    Chunk Name "NARC" (Nitro Archive)                  ;\
  004h 2    Byte Order (FFFEh)                                 ;
  006h 2    Version (0100h)                                    ; NARC
  008h 4    File Size (from "NARC" ID to end of file)          ; Header
  00Ch 2    Chunk Size (0010h)                                 ;
  00Eh 2    Number of following chunks (0003h)                 ;/
  010h 4    Chunk Name "BTAF" (File Allocation Table Block)    ;\
  014h 4    Chunk Size (including above chunk name)            ; File
  018h 2    Number of Files                                    ; Allocation
```

```
  01Ah 2    Reserved (0000h)                                    ; Table
  01Ch ...  FAT (see below)                                     ;/
  ...  4    Chunk Name "BTNF" (File Name Table Block)           ;\
  ...  4    Chunk Size (including above chunk name)             ; File Name
  ...  ...  FNT (see below)                                     ; Table
  ...  ..   Padding for 4-byte alignment (FFh-filled, if any)   ;/
  ...  4    Chunk Name "GMIF" (File Image Block)                ;\
  ...  4    Chunk Size (including above chunk name)             ; File Data
  ...  ...  IMG (File Data)                                     ;/
```

File Allocation Table (FAT) (base/size defined in cart header)
Contains ROM addresses for up to 61440 files (File IDs 0000h and up).

```
  Addr Size Expl.
  00h  4    Start address (originated at IMG base) (0=Unused Entry)
  04h  4    End address   (Start+Len...-1?)        (0=Unused Entry)
```

For NitroROM, addresses must be after Secure Area (at 8000h and up).
For NitroARC, addresses can be anywhere in the IMG area (at 0 and up).
Directories are fully defined in FNT area, and do not require FAT entries.

File Name Table (FNT) (base/size defined in cart header)
Consists of the FNT Directory Table, followed by one or more FNT Sub-Tables.
To interprete the directory tree: Start at the 1st Main-Table entry, which is referencing to a Sub-Table, any directories in the Sub-Table are referencing to Main-Table entries, which are referencing to further Sub-Tables, and so on.

FNT Directory Main-Table (base=FNT+0, size=[FNT+06h]*8)
Consists of a list of up to 4096 directories (Directory IDs F000h and up).

```
  Addr Size Expl.
  00h  4    Offset to Sub-table             (originated at FNT base)
  04h  2    ID of first file in Sub-table   (0000h..EFFFh)
```

For first entry (ID F000h, root directory):

```
  06h  2    Total Number of directories     (1..4096)
```

Further entries (ID F001h..FFFFh, sub-directories):

```
  06h  2    ID of parent directory (F000h..FFFEh)
```

FNT Sub-tables (base=FNT+offset, ends at Type/Length=00h)
Contains ASCII names for all files and sub-directories within a directory.

```
  Addr Size Expl.
  00h  1    Type/Length
              01h..7Fh File Entry          (Length=1..127, without ID field)
              81h..FFh Sub-Directory Entry (Length=1..127, plus ID field)
              00h      End of Sub-Table
              80h      Reserved
  01h  LEN  File or Sub-Directory Name, case-sensitive, without any ending
              zero, ASCII 20h..7Eh, except for characters \/?"<>*:;|
```

Below for Sub-Directory Entries only:

```
  LEN+1 2    Sub-Directory ID (F001h..FFFFh) ;see FNT+(ID AND FFFh)*8
```

File Entries do not have above ID field. Instead, File IDs are assigned in incrementing order (starting at the "First ID" value specified in the Directory Table).

ARM9 and ARM7 Overlay Tables (OVT) (base/size defined in cart header)
Somehow related to Nintendo's compiler, allows to assign compiler Overlay IDs to filesystem File IDs, and to define additional information such like load addresses.

```
  Addr Size Expl.
  00h  4    Overlay ID
  04h  4    RAM Address ;Point at which to load
  08h  4    RAM Size    ;Amount to load
  0Ch  4    BSS Size    ;Size of BSS data region
  10h  4    Static initialiser start address
```

```
  14h  4    Static initialiser end address
  18h  4    File ID  (0000h..EFFFh)
  1Ch  4    Reserved (zero)
```

Cartridge Header
The base/size of FAT, FNT, OVT areas is defined in cartridge header,
DS Cartridge Header

DS Cartridge PassMe/PassThrough

PassMe is an adapter connected between the DS and an original NDS cartridge, used to boot unencrypted code from a flash cartridge in the GBA slot, it replaces the following entries in the original NDS cartridge header:

```
  Addr  Siz Patch
  004h  4   E59FF018h  ;opcode LDR PC,[027FFE24h] at 27FFE04h
  01Fh  1   04h        ;set autostart bit
  022h  1   01h        ;set ARM9 rom offset to nn01nnnnh (above secure area)
  024h  4   027FFE04h  ;patch ARM9 entry address to endless loop
  034h  4   080000C0h  ;patch ARM7 entry address in GBA slot
  15Eh  2   nnnnh      ;adjust header crc16
```

After having verified the encrypted chip IDs (from the original cartridge), the console thinks that it has successfully loaded a NDS cartridge, and then jumps to the (patched) entrypoints.

GBA Flashcard Format
Although the original PassMe requires only the entrypoint, PassMe programs should additionally contain one (or both) of the ID values below, allowing firmware patches to identify & start PassMe games without real PassMe hardware.

```
  0A0h  GBA-style Title    ("DSBooter")
  0ACh  GBA-style Gamecode ("PASS")
  0C0h  ARM7 Entrypoint    (32bit ARM code)
```

Of course, that applies only to early homebrew programs, newer games should use normal NDS cartridge headers.

ARM9 Entrypoint
The GBA-slot access rights in the EXMEMCNT register are initially assigned to the ARM7 CPU, so the ARM9 cannot boot from the flashcard, instead it is switched into an endless loop in Main RAM (which contains a copy of the cartridge header at 27FFE00h and up). The ARM7 must thus copy ARM9 code to Main RAM, and then set the ARM9 entry address by writing to [027FFE24h].

DS Cartridge GBA Slot

Aside from the 17-pin NDS slot, the DS also includes a 32-pin GBA slot. This slot is used for GBA backwards compatibility mode. Additionally, in DS mode, it can be as expansion port, or for importing data from GBA games.

```
  NDS:     Normal 32pin slot
  DS Lite: Short 32pin slot (GBA cards stick out)
  DSi:     N/A (dropped support for GBA carts, and for DS-expansions)
```

In DS mode, ROM, SRAM, FLASH backup, and whatever peripherals contained in older GBA cartridges can be accessed (almost) identically as in GBA mode,
GBA Cartridges

Addressing
In DS mode, only one ROM-region is present at 8000000h-9FFFFFFh (ie. without the GBA's mirrored WS1 and WS2 regions at A000000h-DFFFFFFh). The expansion region (for SRAM/FLASH/etc) has been moved from E000000h-E00FFFFh (GBA-mode) to A000000h-A00FFFFh (DS-mode).

Timings
GBA timings are specified as "waitstates" (excluding 1 access cycle), NDS timings are specified as (total) "access time". And, the NDS bus-clock is twice as fast as for GBA. So, for "N" GBA waitstates, the NDS access time should be "(N+1)*2". Timings are controlled via NDS EXMEMCNT instead GBA WAITCNT,
DS Memory Control - Cartridges and Main RAM

GBA EEPROM
EEPROMs in GBA carts cannot be accessed in DS mode. The EEPROMs should be accessed with 8 waits on GBA, ie. 18 cycles on NDS on both 1st/2nd access. But, 2nd access is restricted to max 6 cycles in NDS mode, which is ways too fast.

DS Cart Rumble Pak

DS Rumble Option Pak
The Rumble Pak comes bundled with Metroid Prime Pinball. It contains a small actuator made by ALPS to make it rumble. The original device (NTR-008) is sized like a normal GBA cartridge, and there's also shorter variant for the DS-Lite (USG-006).
The rumble pak is pretty simple internally, it only wires up to a few pins on the GBA Cartridge Port:
```
VCC, GND, /WR, AD1, and IRQ (grounded)
```
AD1 runs into a little 8 pin chip, which is probably just a latch on the rising edge of /WR. A line runs from this chip to a transistor that is directly connected to the actuator. The only other chip on the board is a 5 pin jobber, probably a power component.
For detection, AD1 seems to be pulled low when reading from it, the other AD lines are open bus (containing the halfword address), so one can do:
```
for i=0 to 0FFFh
  if halfword[8000000h+i*2]<>(i and FFFDh) then <not_a_ds_rumble_pak>
next i
```
The actuator doesn't have an on/off setting like a motor, it rumbles when you switch it between the two settings. Switch frequently for a fast rumble, and fairly slowly for more of a 'tick' effect. That should be done via timer irq:
```
rumble_state = rumble_state xor 0002h
halfword[8000000h]=rumble_state
```
Unknown if one of the two states has higher power-consumption than the other, ie. if it's a "pull/release" mechanism, if so, then disabling rumble should be done by using the "release" state, which would be AD1=0, or AD1=1...?
Note: The v3 firmware can detect the Rumble Pak as an option pak, but it does not provide an enable/disable rumble option in the alarm menu.

Other DS Rumble device
There's also another DS add-on with rumble. That device uses AD8 (instead AD1) to control rumble, and, it's using a classic motor (ie. it's rumbling while and as long as the latched AD8 value is "1").
DS Cart Slider with Rumble

GBA Rumble Carts
There are also a few GBA games that contain built-in Rumble, and which could be used by NDS games as well. To be user friendly, best support both types.
GBA Cart Rumble

DS Cart Slider with Rumble

Add-on device for the japanese title Magukiddo. The optical sensor is attached underneath of the console

(connected to the GBA slot).
The sensor is an Agilent ADNS-2030 Low Power Optical Mouse Sensor (16pin DIP chip with built-in optical sensor, and external LED light source) with two-wire serial bus (CLK and DTA).

ADNS-2030 Registers (write 1 byte index, then read/write 1 byte data)
Index (Bit7=Direction; 0=Read, 1=Write):

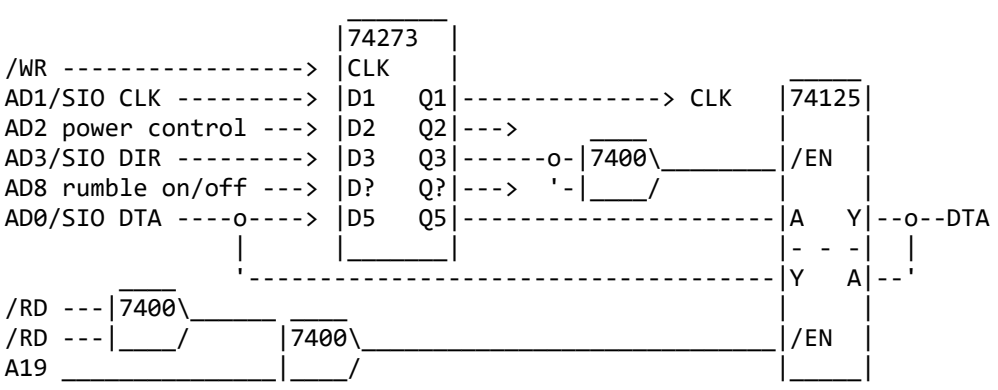
```
  00h Product_ID (R) (03h)
  01h Revision_ID (R) (10h=Rev. 1.0) (20h=Used in DS-option-pak)
  02h Motion/Status Flags (R)
  03h Delta_X (R) (signed 8bit) (automatically reset to 00h after reading)
  04h Delta_Y (R) (signed 8bit) (automatically reset to 00h after reading)
  05h SQUAL (R) (surface quality) (unsigned 8bit)
  06h Average_Pixel (R) (unsigned 6bit, upper 2bit unused)
  07h Maximum_Pixel (R) (unsigned 6bit, upper 2bit unused)
  08h Reserved
  09h Reserved
  0Ah Configuration_bits (R/W)
  0Bh Reserved
  0Ch Data_Out_Lower (R)
  0Dh Data_Out_Upper (R)
  0Eh Shutter_Lower (R)
  0Fh Shutter_Upper (R)
  10h Frame_Period_Lower (R/W)
  11h Frame_Period_Upper (R/W)
```

Motion/Status Flags:

```
  7 Motion since last report or PD (0=None, 1=Motion occurred)
  6 Reserved
  5 LED Fault detected (0=No fault,  1=Fault detected)
  4 Delta Y Overflow (0=No overflow, 1=Overflow occured)
  3 Delta X Overflow (0=No overflow, 1=Overflow occured)
  2 Reserved
  1 Reserved
  0 Resolution in counts per inch (0=400, 1=800)
```

Configuration_bits:

```
  7 Reset Power up defaults (W) (0=No, 1=Reset)
  6 LED Shutter Mode (0=LED always on, 1=LED only on when shutter is open)
  5 Self Test (W) (0=No, 1=Perform all self tests)
  4 Resolution in counts per inch (0=400, 1=800)
  3 Dump 16x16 Pixel bitmap (0=No, 1=Dump via Data_Out ports)
  2 Reserved
  1 Reserved
  0 Sleep Mode (0=Normal/Sleep after 1 second, 1=Always awake)
```



7400 Quad NAND Gate, 74273 8bit Latch

AD0 Optical Sensor Serial Data (0=Low, 1=High)
AD1 Optical Sensor Serial Clock (0=Low, 1=High)
AD2 Optical Sensor Power (0=Off, 1=On)
AD3 Optical Sensor Serial Direction (0=Read, 1=Write)

AD8 Rumble Motor (0=Off, 1=On)

Thanks: Daniel Palmer

DS Cart Expansion RAM

DS Memory Expansions
There are several RAM expansions for the NDS. The RAM cartridge connects to the GBA slot; can can be then accessed from cartridges in the DS slot.

```
Opera         (8MB RAM) (official RAM expansion for Opera browser)
EZ3/4/3-in-1  (8-16MB RAM, plus FLASH, plus rumble)
Supercard     (32MB)
M3            (32MB)
G6            (32MB)
```

The recommended access time (waitstates) for all memory types is unknown. Unknown which programs do use these expansions for which purposes (aside from the Opera browser).
Thanks to Rick "Lick" Wong for info on detection and unlocking.

Opera / DS Memory Expansion Pak (NTR-011 or USG-007)

```
base=9000000h, size=800000h (8MB)
unlock=1, lock=0
STRH [8240000h],lock/unlock
```

EZ

```
base=8400000h, size=VAR (8MB..16MB)
locking/unlocking/detection see below
```

Supercard

```
base=8000000h, size=1FFFFFEh (32MB minus last two bytes?)
unlock=5 (RAM_RW), lock=3 (MEDIA)
STRH [9FFFFFEh],A55Ah
STRH [9FFFFFEh],A55Ah
STRH [9FFFFFEh],lock/unlock
STRH [9FFFFFEh],lock/unlock
```

M3

```
base=8000000h, size=2000000h (32MB)
unlock=00400006h, lock=00400003h
LDRH Rd,[8E00002h]
LDRH Rd,[800000Eh]
LDRH Rd,[8801FFCh]
LDRH Rd,[800104Ah]
LDRH Rd,[8800612h]
LDRH Rd,[8000000h]
LDRH Rd,[8801B66h]
LDRH Rd,[8000000h+(lock/unlock)*2]
LDRH Rd,[800080Eh]
LDRH Rd,[8000000h]
LDRH Rd,[80001E4h]
LDRH Rd,[80001E4h]
LDRH Rd,[8000188h]
LDRH Rd,[8000188h]
```

G6

```
base=8000000h, size=2000000h (32MB)
unlock=6, lock=3
LDRH Rd,[9000000h]
LDRH Rd,[9FFFFE0h]
LDRH Rd,[9FFFFECh]
```

```
  LDRH Rd,[9FFFFECh]
  LDRH Rd,[9FFFFECh]
  LDRH Rd,[9FFFFFCh]
  LDRH Rd,[9FFFFFCh]
  LDRH Rd,[9FFFFFCh]
  LDRH Rd,[9FFFF4Ah]
  LDRH Rd,[9FFFF4Ah]
  LDRH Rd,[9FFFF4Ah]
  LDRH Rd,[9200000h+(lock/unlock)*2]
  LDRH Rd,[9FFFFF0h]
  LDRH Rd,[9FFFFE8h]
```

Detection

For EZ, detection works as so:

```
 ez_ram_test:   ;Based on DSLinux Amadeus' detection
  ez_subfunc(9880000h,8000h) ;-SetRompage (OS mode)
  ez_subfunc(9C40000h,1500h) ;-OpenNorWrite
  [08400000h]=1234h          ;\
  if [08400000h]=1234h       ; test writability at 8400000h
    [8000000h]=4321h         ; and non-writability at 8000000h
    if [8000000h]<>4321h     ;
      return true            ;/
  ez_subfunc(9C40000h,D200h) ;CloseNorWrite
  ez_subfunc(9880000h,0160h) ;SetRompage (0160h)
  ez_subfunc(9C40000h,1500h) ;OpenNorWrite
  [8400000h]=1234h           ;\
  if [8400000h]=1234h        ; test writability at 8400000h
    return true              ;/
  return false               ;-failed
 ez_subfunc(addr,data):
  STRH [9FE0000h],D200h
  STRH [8000000h],1500h
  STRH [8020000h],D200h
  STRH [8040000h],1500h
  STRH [addr],data
  STRH [9FC0000h],1500h
```

For all other types (everything except EZ), simply verify that you can write (when unlocked), and that you can't (when locked).

DS Cart Unknown Extras

DS Cartridges with built-in Infrared Port

Some NDS and DSi games (those with NTR-Ixxx or TWL-Ixxx gamecodes) contain built-in Infrared ports; used to communicate with pedometers.

The IR-port is accessed via certain SPI bus commands; that bus is also shared to access FLASH memory via other commands.

The FLASH chip seems to return a nonsense chip ID (maybe the cartridge is using uncommon FLASH memory, or maybe the ID command is redirected to the IR-port hardware).

The ROM chip does also respond with an uncommon ID (with one special bit set, which is possibly indicating the presenence of the IR-hardware) (maybe the IR-port is contained in the ROM chip, or maybe the SPI-bus sharing is handled inside of the ROM chip.

The IR-related SPI commands are mostly unknown. Except that: command 08h should return 55h (or some other non-FFh value), otherwise the game won't work in emulators; this might be some IR-status byte.

DS Cartridges with NAND memory

Some NDS games (eg. Warioware D.I.Y.) contain NAND memory, this memory contains both the game and save memory (normal NDS games contain separate ROM and FLASH/EEPROM chips for that purposes) (the advantage is that NAND allows more storage than the usual FLASH chips).

The Warioware D.I.Y. PCB is marked "DI X-7 C17-01", and it does contain only one single chip, marked "SAMSUNG 004, KLC2811ANB-P204, NTR-UORE-0".
That NAND chip connects directly to the NDS parallel bus (the serial SPI chipselect is left unconnected).
Unknown how to write to the chip, and unknown if certain regions are write-protected.

DS Cartridges with built-in MicroSD Card Slot
The DS Vision cartridge contains a built-in microSD card slot. Users can download videos from internet (against a few), store the videos on microSD cards, and then view them on the NDS via DS Vision cartridge.
Unknown how the microSD is accessed; via parallel 'ROM' bus and/or via serieal SPI bus; by which commands?
Also unknown if the thing contains built-in video decoder hardware, or if videos are decoded on ARM cpus.

DS Cart Cheat Action Replay DS

The first commercial DS cheat code solution, this device was developed by Datel. It supports swapping out cartridges after loading the AR software. For updating, the user may either manually enter codes or use the included proprietary USB cable that comes with the device. The user has been able to manually update codes since firmware version 1.52.

Action Replay DS Codes

```
ABCD-NNNNNNNN       Game ID ;ASCII Gamecode [00Ch] and CRC32 across [0..1FFh]
00000000 XXXXXXXX   manual hook codes (rarely used) (default is auto hook)
0XXXXXXX YYYYYYYY   word[XXXXXXX+offset] = YYYYYYYY
1XXXXXXX 0000YYYY   half[XXXXXXX+offset] = YYYY
2XXXXXXX 000000YY   byte[XXXXXXX+offset] = YY
3XXXXXXX YYYYYYYY   IF YYYYYYYY > word[XXXXXXX]   ;unsigned    ;\
4XXXXXXX YYYYYYYY   IF YYYYYYYY < word[XXXXXXX]   ;unsigned    ; for v1.54,
5XXXXXXX YYYYYYYY   IF YYYYYYYY = word[XXXXXXX]                ; when X=0,
6XXXXXXX YYYYYYYY   IF YYYYYYYY <> word[XXXXXXX]               ; uses
7XXXXXXX ZZZZYYYY   IF YYYY > ((not ZZZZ) AND half[XXXXXXX])   ; [offset]
8XXXXXXX ZZZZYYYY   IF YYYY < ((not ZZZZ) AND half[XXXXXXX])   ; instead of
9XXXXXXX ZZZZYYYY   IF YYYY = ((not ZZZZ) AND half[XXXXXXX])   ; [XXXXXXX]
AXXXXXXX ZZZZYYYY   IF YYYY <> ((not ZZZZ) AND half[XXXXXXX])  ;/
BXXXXXXX 00000000   offset = word[XXXXXXX+offset]
C0000000 YYYYYYYY   FOR loopcount=0 to YYYYYYYY  ;execute Y+1 times
C4000000 00000000   offset = address of the C4000000 code           ;v1.54
C5000000 XXXXYYYY   counter=counter+1, IF (counter AND YYYY) = XXXX ;v1.54
C6000000 XXXXXXXX   [XXXXXXXX]=offset                               ;v1.54
D0000000 00000000   ENDIF
D1000000 00000000   NEXT loopcount
D2000000 00000000   NEXT loopcount, and then FLUSH everything
D3000000 XXXXXXXX   offset = XXXXXXXX
D4000000 XXXXXXXX   datareg = datareg + XXXXXXXX
D5000000 XXXXXXXX   datareg = XXXXXXXX
D6000000 XXXXXXXX   word[XXXXXXXX+offset]=datareg, offset=offset+4
D7000000 XXXXXXXX   half[XXXXXXXX+offset]=datareg, offset=offset+2
D8000000 XXXXXXXX   byte[XXXXXXXX+offset]=datareg, offset=offset+1
D9000000 XXXXXXXX   datareg = word[XXXXXXXX+offset]
DA000000 XXXXXXXX   datareg = half[XXXXXXXX+offset]
DB000000 XXXXXXXX   datareg = byte[XXXXXXXX+offset] ;bugged on pre-v1.54
DC000000 XXXXXXXX   offset = offset + XXXXXXXX
EXXXXXXX YYYYYYYY   Copy YYYYYYYY parameter bytes to [XXXXXXXX+offset...]
44332211 88776655   parameter bytes 1..8 for above code  (example)
0000AA99 00000000   parameter bytes 9..10 for above code (padded with 00s)
FXXXXXXX YYYYYYYY   Copy YYYYYYYY bytes from [offset..] to [XXXXXXX...]
```

IF/ENDIF can be nested up to 32 times. FOR/NEXT cannot be nested, any FOR statement does forcefully terminate any prior loop. FOR does backup the current IF condidition flags, and NEXT does restore these flags, so ENDIF(s) aren't required inside of the loop. The NEXT+FLUSH command does (after finishing the loop) reset offset=0, datareg=0, and does clear all condition flags, so further ENDIF(s) aren't required after the loop.

Before v1.54, the DB000000 code did accidently set offset=offset+XXXXXXX after execution of the code. For all word/halfword accesses, the address should be aligned accordingly. For the COPY commands, addresses should be aligned by four (all data is copied with ldr/str, except, on odd lengths, the last 1..3 bytes do use ldrb/strb).
offset, datareg, loopcount, and counter are internal registers in the action replay software.

> The condition register is checked, for all code types
> but the D0, D1 and D2 code type
Makes sense.

> and for the C5 code type it's checked AFTER the counter has
> been incremented (so the counter is always incremented
I love that exceptions ;-)

Hook
The hook codes consist of a series of nine 00000000 XXXXXXXX codes, and must be marked as (M) code (for not being confused with normal 0XXXXXXX YYYYYYYY codes). For all nine codes, the left 32bit are actually don't care (but should be zero), the meaning of the right 32bit varies from 1st to 9th code.

```
1st: Address used prior to launching game (eg. 23xxxxh)
2nd: Address to write the hook at (inside the ARM7 executable)
3rd: Hook final address (huh?)
4th: Hook mode selection (0=auto, 1=mode1, 2=mode2)
5th: Opcode that replaces the hooked one (eg. E51DE004h)
6th: Address to store important stuff  (default 23FE000h)
7th: Address to store the code handler (default 23FE074h)
8th: Address to store the code list    (default 23FE564h)
9th: Must be 1 (00000001h)
```

For most games, the AR does automatically hook code on the ARM7. Doing that automatically is nice, but hooking ARM7 means that there is no access to VRAM, TCM and Cache, which <might> cause problems since efficient games <should> store all important data in TCM or Cache (though, in practice, I'd doubt that any existing NDS games are that efficient).

Thanks
To Kenobi and Dualscreenman from Kodewerx for above ARDS cheat info.

DS Cart Cheat Codebreaker DS

This is Pelican's entry into the DS cheat-device industry. It supports swapping out the cartridges, and alternately, also gives the user the option of connecting another gamecard onto it. For updating, the user may either manually enter codes, or use Wifi to connect to the Codebreaker update site (that updating will overwrite all manually entered codes though).

Codebreaker DS Codes

```
---Initialization---
0000CR16 GAMECODE                   Specify Game ID, use Encrypted codes
8000CR16 GAMECODE                   Specify Game ID, use Unencrypted codes
BEEFC0DE XXXXXXXX                   Change Encryption Keys
A0XXXXXX YYYYYYYY                   Bootup-Hook 1, X=Address, Y=Value
A8XXXXXX YYYYYYYY                   Bootup-Hook 2, X=Address, Y=Value
F0XXXXXX TYYYYYYY         Code-Hook 1 (T=Type,Y=CheatEngineAddr,X=HookAddr)
F8XXXXXX TPPPPPPP         Code-Hook 2 (T=Type,X=CheatEngineHookAddr,P=Params)
---General codes---
00XXXXXX 000000YY                   [X]=YY
10XXXXXX 0000YYYY                   [X]=YYYY
20XXXXXX YYYYYYYY                   [X]=YYYYYYYY
60XXXXXX 000000YY ZZZZZZZZ 00000000 [[X]+Z]=YY
```

```
60XXXXXX 0000YYYY ZZZZZZZZ 10000000  [[X]+Z]=YYYY
60XXXXXX YYYYYYYY ZZZZZZZZ 20000000  [[X]+Z]=YYYYYYYY
30XXXXXX 000000YY                    [X]=[X] + YY
30XXXXXX 0001YYYY                    [X]=[X] + YYYY
38XXXXXX YYYYYYYY                    [X]=[X] + YYYYYYYY
70XXXXXX 000000YY                    [X]=[X] OR  YY
70XXXXXX 001000YY                    [X]=[X] AND YY
70XXXXXX 002000YY                    [X]=[X] XOR YY
70XXXXXX 0001YYYY                    [X]=[X] OR  YYYY
70XXXXXX 0011YYYY                    [X]=[X] AND YYYY
70XXXXXX 0021YYYY                    [X]=[X] XOR YYYY
---Memory fill/copy---
40XXXXXX 2NUMSTEP 000000YY 000000ZZ  byte[X+(0..NUM-1)*STEP*1]=Y+(0..NUM-1)*Z
40XXXXXX 1NUMSTEP 0000YYYY 0000ZZZZ  half[X+(0..NUM-1)*STEP*2]=Y+(0..NUM-1)*Z
40XXXXXX 0NUMSTEP YYYYYYYY ZZZZZZZZ  word[X+(0..NUM-1)*STEP*4]=Y+(0..NUM-1)*Z
50XXXXXX YYYYYYYY ZZZZZZZZ 00000000  copy Y bytes from [X] to [Z]
---Conditional codes (bugged)---
60XXXXXX 000000YY ZZZZZZZZ 01c100VV  IF [[X]+Z] .. VV   THEN [[X]+Z]=YY
60XXXXXX 000000YY ZZZZZZZZ 01c0VVVV  IF [[X]+Z] .. VVVV THEN [[X]+Z]=YY
60XXXXXX 0000YYYY ZZZZZZZZ 11c100VV  IF [[X]+Z] .. VV   THEN [[X]+Z]=YYYY
60XXXXXX 0000YYYY ZZZZZZZZ 11c0VVVV  IF [[X]+Z] .. VVVV THEN [[X]+Z]=YYYY
60XXXXXX YYYYYYYY ZZZZZZZZ 21c100VV  IF [[X]+Z] .. VV   THEN [[X]+Z]=YYYYYYYY
60XXXXXX YYYYYYYY ZZZZZZZZ 21c0VVVV  IF [[X]+Z] .. VVVV THEN [[X]+Z]=YYYYYYYY
---Conditional codes (working)---
D0XXXXXX NNc100YY                    IF [X] .. YY   THEN exec max(1,NN) lines
D0XXXXXX NNc0YYYY                    IF [X] .. YYYY THEN exec max(1,NN) lines
```

The condition digits (c=0..7), have the following functions:

```
0 IF [mem] =  imm THEN ...              4 IF ([mem] AND imm) =  0   THEN ...
1 IF [mem] <> imm THEN ...              5 IF ([mem] AND imm) <> 0   THEN ...
2 IF [mem] <  imm THEN ... (unsigned)   6 IF ([mem] AND imm) =  imm THEN ...
3 IF [mem] >  imm THEN ... (unsigned)   7 IF ([mem] AND imm) <> imm THEN ...
```

Notes

```
GAMECODE  Cartridge Header[00Ch] (32bit in reversed byte-order)
CR16      Cartridge Header[15Eh] (16bit in normal byte-order)
XXXXXX    27bit addr (actually 7 digits, XXXXXXX, overlaps 5bit code number)
```

The "bugged" conditional codes (60XXXXXX) are accidently skipping NN lines when the condition is false, where NN is taken from the upper 8bit of the code's last 32bit values (ie. exactly as for the D0XXXXXX codes). For byte-writes, that would be NN=01h, which can be eventually dealt with, although there may be compatibility problems which future versions that might fix that bug. For halfword/word writes, NN would be 11h or 21h, so that codes are about totally unusable.

Codebreaker DS / Encrypted Codes

The overall "address value" decryption works like so:

```
for i=4Fh to 00h
  y=77628ECFh
  if i>13h then y=59E5DC8Ah
  if i>27h then y=054A7818h
  if i>3Bh then y=B1BF0855h
  address = (Key0-value) xor address
  value   = value - Key1 - (address ror 1Bh)
  address = (address xor (value + y)) ror 13h
  if (i>13h) then
    if (i<=27h) or (i>3Bh) then x=Key2 xor Key1 xor Key0
    else x=((Key2 xor Key1) and Key0) xor (Key1 and Key2)
    value=value xor (x+y+address)
    x = Secure[((i*4+00h) and FCh)+000h]
    x = Secure[((i*4+34h) and FCh)+100h] xor x
    x = Secure[((i*4+20h) and FCh)+200h] xor x
    x = Secure[((i*4+08h) and FCh)+300h] xor x
    address = address - (x ror 19h)
next i
```

Upon startup, the initial key settings are:

```
Secure[0..7FFh] = Copy of the ENCRYPTED 1st 2Kbytes of the game's Secure Area
```

```
  Key0 = 0C2EAB3Eh, Key1 = E2AE295Dh, Key2 = E1ACC3FFh, Key3 = 70D3AF46h
  scramble_keys
```
Upon BEEFC0DE XXXXXXXX, the keys get changed like so:
```
  Key0 = Key0 + (XXXXXXXX ror 1Dh)
  Key1 = Key1 - (XXXXXXXX ror 05h)
  Key2 = Key2 xor (Key3 xor Key0)
  Key3 = Key3 xor (Key2  -  Key1)
  scramble_keys
```
The above scramble_keys function works like so:
```
  for i=0 to FFh
    y = byte(xlat_table[i])
    Secure[i*4+000h] = (Secure[i*4+000h] xor Secure[y*4]) + Secure[y*4+100h]
    Secure[i*4+400h] = (Secure[i*4+400h] xor Secure[y*4]) - Secure[y*4+200h]
  next i
  for i=0 to 63h
    Key0 = Key0 xor (Secure[i*4] + Secure[i*4+190h])
    Key1 = Key1 xor (Secure[i*4] + Secure[i*4+320h])
    Key2 = Key2 xor (Secure[i*4] + Secure[i*4+4B0h])
    Key3 = Key3 xor (Secure[i*4] + Secure[i*4+640h])
  next i
  Key0 = Key0  -  Secure[7D0h]
  Key1 = Key1 xor Secure[7E0h]
  Key2 = Key2  +  Secure[7F0h]
  Key3 = Key3 xor Secure[7D0h] xor Secure[7F0h]
```
the xlat_table consists of 256 fixed 8bit values:
```
  34h,59h,00h,32h,7Bh,D3h,32h,C9h,9Bh,77h,75h,44h,E0h,73h,46h,06h
  0Bh,88h,B3h,3Eh,ACh,F2h,BAh,FBh,2Bh,56h,FEh,7Ah,90h,F7h,8Dh,BCh
  8Bh,86h,9Ch,89h,00h,19h,CDh,4Ch,54h,30h,01h,93h,30h,01h,FCh,36h
  4Dh,9Fh,FDh,D7h,32h,94h,AEh,BCh,2Bh,61h,DFh,B3h,44h,EAh,8Bh,A3h
  2Bh,53h,33h,54h,42h,27h,21h,DFh,A9h,DDh,C0h,35h,58h,EFh,8Bh,33h
  B4h,D3h,1Bh,C7h,93h,AEh,32h,30h,F1h,CDh,A8h,8Ah,47h,8Ch,70h,0Ch
  17h,4Eh,0Eh,A2h,85h,0Dh,6Eh,37h,4Ch,39h,1Fh,44h,98h,26h,D8h,A1h
  B6h,54h,F3h,AFh,98h,83h,74h,0Eh,13h,6Eh,F4h,F7h,86h,80h,ECh,8Eh
  EEh,4Ah,05h,A1h,F1h,EAh,B4h,D6h,B8h,65h,8Ah,39h,B3h,59h,11h,20h
  B6h,BBh,4Dh,88h,68h,24h,12h,9Bh,59h,38h,06h,FAh,15h,1Dh,40h,F0h
  01h,77h,57h,F5h,5Dh,76h,E5h,F1h,51h,7Dh,B4h,FAh,7Eh,D6h,32h,4Fh
  0Eh,C8h,61h,C1h,EEh,FBh,2Ah,FCh,ABh,EAh,97h,D5h,5Dh,E8h,FAh,2Ch
  06h,CCh,86h,D2h,8Ch,10h,D7h,4Ah,CEh,8Fh,EBh,03h,16h,ADh,84h,98h
  F5h,88h,2Ah,18h,ACh,7Fh,F6h,94h,FBh,3Fh,00h,B6h,32h,A2h,ABh,28h
  64h,5Ch,0Fh,C6h,23h,12h,0Ch,D2h,BAh,4Dh,A3h,F2h,C9h,86h,31h,57h
  0Eh,F8h,ECh,E1h,A0h,9Ah,3Ch,65h,17h,18h,A0h,81h,D0h,DBh,D5h,AEh
```
all used operations are unsigned 32bit integer.

Thanks
To Kenobi and Dualscreenman from Kodewerx for above CBDS cheat info.

DS Cart DLDI Driver

DLDI (Dynamically Linked Device Interface for libfat) is a popular yet undocumented flashcart driver for homebrew NDS software dating back to 2006. Below was reverse-engineered 11/2018.

file.dldi --> driver file (can be small like 1.5Kbyte, or max 32Kbyte)
file.nds --> ROM image (must contain 32Kbyte space with DLDI ID for patching)

Driver patch file standard header
```
  00h 4  DLDI ID       (EDh,A5h,8Dh,BFh) (aka BF8DA5EDh)  ;\patching tools will
  04h 8  DLDI String   (20h,"Chishm",00h)                 ; refuse any other
  0Ch 1  DLDI Version  (01h in .dldi, don't care in .nds) ;/values
  0Dh 1  Size of .dldi+BSS (rounded up to 1 SHL N bytes) (max 0Fh=32Kbytes)
  0Eh 1  Sections to fix/destroy (see FIX_xxx)
```

```
  0Fh 1  Space in .nds file (1 SHL N) (0Eh..0Fh in .nds, can be 0 in .dldi)
  10h 48 ASCII Full Driver Name (max 47 chars, plus zero padding)
  40h 4  Address of ALL start (text) ;-base address (BF800000h in .dldi)
  44h 4  Address of ALL end   (data) ;-for highly-unstable FIX_ALL addr.adjusts
  48h 4  Address of GLUE start       ;\for semi-stable FIX_GLUE addr.adjusts
  4Ch 4  Address of GLUE end         ;/  ("Interworking glue" for ARM-vs-THUMB)
  50h 4  Address of GOT start        ;\for semi-stable FIX_GOT addr.adjusts
  54h 4  Address of GOT end          ;/  ("Global Offset Table")
  58h 4  Address of BSS start        ;\for zerofilling "BSS" via FIX_BSS
  5Ch 4  Address of BSS end          ;/  ("Block Started by Symbol")
  60h 4  ASCII Short Driver/Device Name (4 chars, eg. "MYHW" for MyHardware)
  64h 4  Flags 2 (see FEATURE_xxx) (usually 13h=GbaSlot, or 23h=NdsSlot)
  68h 4  Address of Function startup() ;<-- must be at offset +80h !! ;\
  6Ch 4  Address of Function isInserted() ;out: 0=no/fail, 1=yes/okay ; all
  70h 4  Address of Function readSectors(sector,numSectors,buf)       ; return
  74h 4  Address of Function writeSectors(sector,numSectors,buf)      ; 0=fail,
  78h 4  Address of Function clearStatus()                            ; 1=okay
  7Ch 4  Address of Function shutdown()                               ;/
  80h .. Driver Code (can/must begin with "startup()")            ;\max 7F80h
  ..  .. Glue section (usually a small snippet within above code) ; bytes (when
  ..  .. GOT section (usually after above code) (pointer table)   ; having 32K
  ..  .. BSS section (usually at end, may exceed .dldi filesize)  ; allocated)
  ..  .. Optional two garbage NOPs at end of default.dldi         ;/
```

hdr[0Eh] - Sections to fix/destroy (FIX_xxx):

```
  0    FIX_ALL   ;-installer uses highly-unstable guessing in whole dldi file
  1    FIX_GLUE  ;-installer uses semi-stable address guessing in GLUE area
  2    FIX_GOT   ;-installer uses semi-stable address guessing in GOT area
  3    FIX_BSS   ;-installer will zerofill BSS area
  4-7  Reserved (0)
```

hdr[64h] - Flags 2 (FEATURE_xxx) (usually 13h=GbaSlot, or 23h=NdsSlot):

```
  0    FEATURE_MEDIUM_CANREAD          00000001h (usually set)
  1    FEATURE_MEDIUM_CANWRITE         00000002h (a few carts can't write)
  2-3  Reserved (0)
  4    FEATURE_SLOT_GBA                00000010h (need EXMEMCNT bit7 adjusted)
  5    FEATURE_SLOT_NDS                00000020h (need EXMEMCNT bit11 adjusted)
  6-31 Reserved (0)
```

Note: The allocated driver size in hdr[0Fh] was 0Fh=32Kbytes between 2006 and 2016, however, libnds has changed that to 0Eh=16Kbytes in January 2017 (maybe intending to free more RAM, especially when using ARM7 WRAM).
However, there's at least one driver exceeding 16K (rpg_nand.dldi in AKAIO package; the driver disguises itself as 8K driver in hdr[0Dh], but its BSS area actually needs ways MORE than 16K).

Required entries in .nds file

Officially, dldi could be at any 4-byte aligned location, however, for faster lookup, better use this locations:

```
  dldi area should be located at a 40h-byte aligned address in ROM image.
  dldi area should be located in ARM9 (or ARM7) bootcode area.
```

An "empty" driver needs to contain:

```
  dldi[00h..0Bh] must contain DLDI ID word/string
  dldi[0Fh]      must contain allocated size (0Eh=16Kbyte or 0Fh=32Kbyte)
  dldi[40h..43h] must contain RAM base address of DLDI block
  and other entries should contain valid dummy strings and dummy functions.
```

An installed driver should contain a copy of the .dldi file, with addresses adjusted to RAM locations, and BSS area zerofilled (if FIX_BSS was set)

```
  dldi[0Fh] must be kept as in the old .nds file (not as in .dldi file)
```

Some installers might try to detect homebrew by looking at nds carthdr for deciding whether or not to try to install dldi (unknown if/which ones are doing such things and looking at which carthdr entries).

Functions

startup, isInserted, clearStatus, shutdown can be dummy functions that do nothing (other than returning r0=1=okay).
Alternately startup/shutdown can initialize or power down the hardware, clearStatus is meant to be some sort of

soft reset, and isInserted is allowing to test if the SD card is inserted & working.
read/write sectors are reading/writing one or more sectors. Sector size is 200h bytes, sector numbers is 0=First 200h bytes, 1=Next 200h bytes, and so on.
buf should be usually 4-byte aligned, however, some drivers do also support unaligned buffers using slower transfer code (better implement that when making .dldi drivers, but better don't rely on it being supported when making .nds files).
The driver functions can support SD and SDHC (or the flashcart manufacturer might release driver updates if SDHC wasn't supported).
Higher level FAT functions must be contained in the .nds file (so a driver update won't help if the .nds file lacks FAT32) (and ExFAT most unlikely to be supported).
Functions should be ARM7 compatible, ie. don't use BLX or POP r15 for mode switching, so the driver can be used on both ARM9 and ARM7 (or even on GBA).

SLOT_GBA/NDS
SLOT_GBA/NDS seem to relate to GBA and NDS slots, the driver can probably have only one of the SLOT bits set (the functions don't allow to select which slot to use).
Purpose is unclear to me, maybe just telling the .nds file that the flashcart is in the given slot (and thereby shouldn't expect other hardware in that slot). Or maybe telling telling the installer which hardware the driver is supposed for.

FIX_xxx does maybe relate to address adjustments made by the dldi installer. Unknown if/how that's working.
DS Cart DLDI Driver - Guessed Address-Adjustments

Some DLDI flashcarts support extra features like Rumble. However, that extra hardware is accessed via direct I/O, not via DLDI driver. Unknown which I/O ports are used for that stuff - probably something compatible with official GBA/NDS rumble cart(s).

DS Cart DLDI Driver - Guessed Address-Adjustments

The DLDI installer uses some guessing method for address-adjustments (the FIX_xxx flags are supposed to patch addresses, but not opcodes or other data).

Unaligned-Word patching and over-shooting bug
The central mistake in the official DLDI installer is that it is patching all words at [start..end-1], using 1-byte address increments instead of 4-byte increments. This includes patching words at non-word aligned locations, or patching words whose lower bytes were already patched, and over-shooting to words at [end-1..end+2].

ddmem Base BF800000h
Most or all .dldi files are using ddmem base BF800000h (defined in dldi.ld). That value was chosen because it won't conflict with opcodes (as NDS BIOS doesn't use SWI function 80h, BF80xxxxh would be an invalid SWI function in ARM; and BF80h would be an invalid opcode in THUMB).
So far, this would have worked well, but it doesn't work with unaligned-word patching bug (eg. THUMB opcodes 8000h,BF00h, or ARM opcodes xxBF80xxh or the like). And, even if it would have worked for opcodes - it might still fail for data values.

FIX_GOT
GOT does usually contain BF80xxxxh address pointers (plus some 00000000h words). The guessing works quite stable (the maximum for 32K files is xxxxh=7FFFh, so there's no risk to encounter xxxx=BF80h) (one could encounter xxxxh=xxBFh, but the previously patched word is usually in RAM area, eg. 02xxxxxxh, so this would form BF02h, without risk to be seen as BF80h).
BUG: The GOT table is usually located at the end of the .dldi file, meaning that the over-shooting bug will see three uninitialized bytes at [got_end+0..2], and may go amok if they are BF80xxh or xxBF80h. The value of

those bytes depends on left-over from previously installed .dldi driver(s) and on the ddmem base used in the .nds file, so the bug may take place randomly depending on several factors.

FIX_GLUE
GLUE does usually contain a handful of opcodes and .pool values for switching between ARM and THUMB code. The intention is to patch the addresses in the .pool, and to leave the opcodes intact. This can be potentially stable, assuming that the used opcodes in the GLUE (and the next three bytes after glue_end) usually won't contain BF80h).

FIX_ALL
This is the mother of all bugs. Fortunately there aren't any .dldi drivers with FIX_ALL flag - and one should never make drivers that do use it.
ALL is covering the whole dldi space, including the 80h-byte DLDI header, the code area, including GLUE area, and GOT area, and probably also the yet uninitialized BSS area, and the next three bytes after end_all.
Patching the whole code area means an increased risk to hit opcodes or data values that contain BF80h. The over-shooting bug may even destroy the next three bytes after the 32Kbyte area.
Patching the DLDI header could destroy the header itself, the header in the .dldi file usually won't contain BF80h at unintended locations, however, the pointers in that header are adjusted before applying FIX_ALL, for example, RAM base 0200BF00h (in .nds file) combined with a function at BF800080h (in .dldi file) would result in 0200BF80h. The nasty thing is that the problem won't occur with other RAM base values (in other .nds files).

Avoiding the bugs
When making .dldi drivers: Never use FIX_ALL. And preferably avoid FIX_GOT and FIX_GLUE as well (ARM CPU can do relative jumps and relative addressing, eg. via ADR and ADRL pseudo opcodes, so there's no point in needing address adjustments). Or otherwise append padding after GOT area, and try to avoid using opcodes/data with BF80h in/after GLUE area.
When making dldi installers: Best patch only word-aligned words (ARM CPU can't access unaligned data, so there's little chance that DLDI drivers would ever contain unaligned words). Or, when maintaining unaligned patching: At the very least skip the 80h-byte header on FIX_ALL, and after patching a word at other locations, skip the next three bytes, and don't do the over-shooting at end.
When making .nds files: There isn't too much that could be done here. One could set ddmem to 64Kbyte aligned addresses (so functions won't end up at xxxxBF80h). Or one could even set ddmem to BF800000h (so patching will leave everything unchanged & intact - so one could then do the address adjustments manually, and hopefully more reliable than other DLDI installers).

DS Encryption by Gamecode/Idcode (KEY1)

KEY1 - Gamecode / Idcode Encryption
The KEY1 encryption relies only on the gamecode (or firmware idcode), it does not contain any random components. The fact that KEY1 encrypted commands appear random is just because the <unencrypted> commands contain random values, so the encryption result looks random.

KEY1 encryption is used for KEY1 encrypted gamecart commands (ie. for loading the secure area). It is also used for resolving the extra decryption of the first 2K of the secure area, and for firmware decryption, and to decode some encrypted values in gamecart/firmware header.

Initial Encryption Values
Below formulas can be used only with a copy of the 1048h-byte key tables from NDS/DSi BIOS. The values can be found at:
```
NDS.ARM7 ROM: 00000030h..00001077h (values 99 D5 20 5F ..) Blowfish/NDS-mode
DSi.ARM9 ROM: FFFF99A0h..FFFFA9E7h (values 99 D5 20 5F ..) ""
DSi.TCM Copy: 01FFC894h..01FFD8DBh (values 99 D5 20 5F ..) ""
```

```
DSi.ARM7 ROM: 0000C6D0h..0000D717h (values 59 AA 56 8E ..) Blowfish/DSi-mode
DSi.RAM Copy: 03FFC654h..03FFD69Bh (values 59 AA 56 8E ..) ""
DSi.Debug:    (stored in launcher) (values 69 63 52 05 ..) Blowfish/DSi-debug
```

The DSi ROM sections are disabled after booting, but the RAM/TCM copies can be dumped (eg. with some complex main memory hardware mods, or via unlaunch exploit). The DSi.Debug key is stored in launcher, and it's used when SCFG_OP is nonzero (as so on debugging on hardware).

encrypt_64bit(ptr) / decrypt_64bit(ptr)

```
Y=[ptr+0]
X=[ptr+4]
FOR I=0 TO 0Fh (encrypt), or FOR I=11h TO 02h (decrypt)
  Z=[keybuf+I*4] XOR X
  X=[keybuf+048h+((Z SHR 24) AND FFh)*4]
  X=[keybuf+448h+((Z SHR 16) AND FFh)*4] + X
  X=[keybuf+848h+((Z SHR  8) AND FFh)*4] XOR X
  X=[keybuf+C48h+((Z SHR  0) AND FFh)*4] + X
  X=Y XOR X
  Y=Z
NEXT I
[ptr+0]=X XOR [keybuf+40h] (encrypt), or [ptr+0]=X XOR [keybuf+4h] (decrypt)
[ptr+4]=Y XOR [keybuf+44h] (encrypt), or [ptr+4]=Y XOR [keybuf+0h] (decrypt)
```

apply_keycode(modulo)

```
encrypt_64bit(keycode+4)
encrypt_64bit(keycode+0)
[scratch]=0000000000000000h   ;S=0 (64bit)
FOR I=0 TO 44h STEP 4         ;xor with reversed byte-order (bswap)
  [keybuf+I]=[keybuf+I] XOR bswap_32bit([keycode+(I MOD modulo)])
NEXT I
FOR I=0 TO 1040h STEP 8
  encrypt_64bit(scratch)      ;encrypt S (64bit) by keybuf
  [keybuf+I+0]=[scratch+4]    ;write S to keybuf (first upper 32bit)
  [keybuf+I+4]=[scratch+0]    ;write S to keybuf (then lower 32bit)
NEXT I
```

init_keycode(idcode,level,modulo,key)

```
if key=nds then copy [nds_arm7bios+0030h..1077h] to [keybuf+0..1047h]
if key=dsi then copy [dsi_arm7bios+C6D0h..D717h] to [keybuf+0..1047h]
[keycode+0]=[idcode]
[keycode+4]=[idcode]/2
[keycode+8]=[idcode]*2
IF level>=1 THEN apply_keycode(modulo) ;first apply (always)
IF level>=2 THEN apply_keycode(modulo) ;second apply (optional)
[keycode+4]=[keycode+4]*2
[keycode+8]=[keycode+8]/2
IF level>=3 THEN apply_keycode(modulo) ;third apply (optional)
```

firmware_decryption

```
init_keycode(firmware_header+08h,1,0Ch,nds) ;idcode (usually "MACP"), level 1
decrypt_64bit(firmware_header+18h)          ;rominfo
init_keycode(firmware_header+08h,2,0Ch,nds) ;idcode (usually "MACP"), level 2
decrypt ARM9 and ARM7 bootcode by decrypt_64bit (each 8 bytes)
decompress ARM9 and ARM7 bootcode by LZ77 function (swi)
calc CRC16 on decrypted/decompressed ARM9 bootcode followed by ARM7 bootcode
```

Note: The sizes of the compressed/encrypted bootcode areas are unknown (until they are fully decompressed), one way to solve that problem is to decrypt the next 8 bytes each time when the decompression function requires more data.

gamecart_decryption

```
init_keycode(cart_header+0Ch,1,08h,nds)   ;gamecode, level 1, modulo 8
decrypt_64bit(cart_header+78h)            ;rominfo (secure area disable)
init_keycode(cart_header+0Ch,2,08h,nds)   ;gamecode, level 2, modulo 8
```

```
encrypt_64bit all NDS KEY1 commands (1st command byte in MSB of 64bit value)
after loading the secure_area, calculate secure_area crc, then
decrypt_64bit(secure_area+0)              ;first 8 bytes of secure area
init_keycode(cart_header+0Ch,3,08h,nds)   ;gamecode, level 3, modulo 8
decrypt_64bit(secure_area+0..7F8h)        ;each 8 bytes in first 2K of secure
init_keycode(cart_header+0Ch,1,08h,dsi)   ;gamecode, level 1, modulo 8
encrypt_64bit all DSi KEY1 commands (1st command byte in MSB of 64bit value)
```

After secure area decryption, the ID field in the first 8 bytes should be "encryObj", if it matches then first 8 bytes are filled with E7FFDEFFh, otherwise the whole 2K are filled by that value.

Gamecart Command Register

Observe that the byte-order of the command register [40001A8h] is reversed. The way how the CPU stores 64bit data in memory (and the way how the "encrypt_64bit" function for KEY1-encrypted commands expects data in memory) is LSB at [addr+0] and MSB at [addr+7]. This value is to be transferred MSB first. However, the DS hardware transfers [40001A8h+0] first, and [40001A8h+7] last. So, the byte order must be reversed when copying the value from memory to the command register.

Note

The KEY1 encryption is based on Bruce Schneier's "Blowfish Encryption Algorithm".

DS Encryption by Random Seed (KEY2)

KEY2 39bit Seed Values

The pre-initialization settings at cartridge-side (after reset) are:

```
Seed0 = 58C56DE0E8h
Seed1 = 5C879B9B05h
```

The post-initialization settings (after sending command 4llllmmmnnnkkkkkh to the cartridge, and after writing the Seed values to Port 40001Bxh) are:

```
Seed0 = (mmmnnn SHL 15)+6000h+Seedbyte
Seed1 = 5C879B9B05h
```

The seedbyte is selected by Cartridge Header [013h].Bit0-2, this index value should be usually in range 0..5, however, possible values for index 0..7 are: E8h,4Dh,5Ah,B1h,17h,8Fh,99h,D5h.
The 24bit random value (mmmnnn) is derived from the real time clock setting, and also scattered by KEY1 encryption, anyways, it's just random and doesn't really matter where it comes from.

KEY2 Encryption

Relies on two 39bit registers (x and y), which are initialized as such:

```
x = reversed_bit_order(seed0)  ;ie. LSB(bit0) exchanged with MSB(bit38), etc.
y = reversed_bit_order(seed1)
```

During transfer, x, y, and transferred data are modified as such:

```
x = (((x shr 5)xor(x shr 17)xor(x shr 18)xor(x shr 31)) and 0FFh)+(x shl 8)
y = (((y shr 5)xor(y shr 23)xor(y shr 18)xor(y shr 31)) and 0FFh)+(y shl 8)
data = (data xor x xor y) and 0FFh
```

DS Firmware Serial Flash Memory

ST Microelectronics SPI Bus Compatible Serial FLASH Memory

```
Chips used as wifi-flash:
 ID 20h,40h,12h - ST M45PE20 - 256 KBytes (Nintendo DS) (in my old DS)
 ID 20h,50h,12h - ST M35PE20 - 256 KBytes (Nintendo DS) (in my DS-Lite)
 ID 20h,80h,13h - ST M25PE40 - 512 KBytes (iQue DS, with chinese charset)
 ID 20h,40h,11h - ST 45PE10V6 - 128 Kbytes (Nintendo DSi) (in my DSi)
 ID 20h,58h,0Ch?- 5A32        - 4 Kbytes (Nintendo DSi, newer models)
 ID ?           - 26FV032T    - (Nintendo DSi, J27H020) (this has big package)
 ID ?           - 5K32        - (3DS?)
```

```
  ID 62h,62h,0Ch - 32B, 3XH    - 4 Kbytes (New3DS)
 Other similar chips (used in game cartridges):
  ID 20h,40h,13h - ST 45PE40V6 - 512 KBytes (DS Zelda, NTR-AZEP-0)
  ID 20h,40h,14h - ST 45PE80V6 - 1024 Kbytes (eg. Spirit Tracks, NTR-BKIP)
 +ID 62h,11h,00h - Sanyo ?          - 512 Kbytes (P-Letter Diamond, ADAE)
  ID 62h,16h,00h - Sanyo LE25FW203T - 256 KBytes (Mariokart backup)
 +ID 62h,26h,11h - Sanyo ?          - ? Kbytes (3DS: CTR-P-AXXJ)
 +ID 62h,26h,13h - Sanyo ?          - ? Kbytes (3DS: CTR-P-APDJ)
  ID C2h,22h,11h - Macronix MX25L1021E? 128 Kbytes (eg. 3DS Starfox)
  ID C2h,22h,13h - Macronix ...? 512 Kbytes (eg. 3DS Kid Icarus, 3DS Sims 3)
  ID C2h,20h,17h - Macronix MX25L6445EZNI-10G 8192 Kbytes (DSi Art Academy)
  ID 01h,F0h,00h - Garbage/Infrared on SPI-bus? (eg. P-Letter White)
  ID 03h,F8h,00h - Garbage/Infrared on SPI-bus? (eg. P-Letter White 2)
```

FLASH has more than 100,000 Write Cycles, more than 20 Year Data Retention
The Firmware Flash Memory is accessed via SPI bus,
DS Serial Peripheral Interface Bus (SPI)

Instruction Codes

```
  06h  WREN Write Enable (No Parameters)
  04h  WRDI Write Disable (No Parameters)
  9Fh  RDID Read JEDEC Identification (Read 1..3 ID Bytes)
            (Manufacturer, Device Type, Capacity)
  05h  RDSR Read Status Register (Read Status Register, endless repeated)
            Bit7-2  Not used (zero)
            Bit1    WEL Write Enable Latch             (0=No, 1=Enable)
            Bit0    WIP Write/Program/Erase in Progess (0=No, 1=Busy)
  03h  READ Read Data Bytes (Write 3-Byte-Address, read endless data stream)
  0Bh  FAST Read Data Bytes at Higher Speed (Write 3-Byte-Address, write 1
            dummy-byte, read endless data stream) (max 25Mbit/s)
  0Ah  PW   Page Write (Write 3-Byte-Address, write 1..256 data bytes)
            (changing bits to 0 or 1) (reads unchanged data, erases the page,
            then writes new & unchanged data) (11ms typ, 25ms max)
  02h  PP   Page Program (Write 3-Byte-Address, write 1..256 data bytes)
            (changing bits from 1 to 0) (1.2ms typ, 5ms max)
  DBh  PE   Page Erase 100h bytes (Write 3-Byte-Address) (10ms typ, 20ms max)
  D8h  SE   Sector Erase 10000h bytes (Write 3-Byte-Address) (1s typ, 5s max)
  B9h  DP   Deep Power-down (No Parameters) (consumption 1uA typ, 10uA max)
            (3us) (ignores all further instructions, except RDP)
  ABh  RDP  Release from Deep Power-down (No Parameters) (30us)
```

Write/Program may not cross page-boundaries. Write/Program/Erase are rejected during first 1..10ms after power up. The WEL bit is automatically cleared on Power-Up, on /Reset, and on completion of WRDI/PW/PP/PE/SE instructions. WEL is set by WREN instruction (which must be issued before any write/program/erase instructions). Don't know how RDSR behaves when trying to write to the write-protected region?

Communication Protocol

```
  Set Chip Select LOW to invoke the command
  Transmit the instruction byte
  Transmit any parameter bytes
  Transmit/receive any data bytes
  Set Chip Select HIGH to finish the command
```

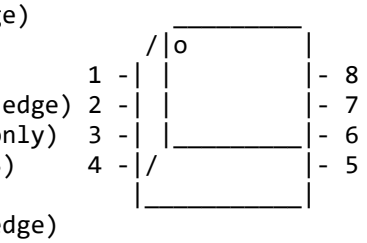
All bytes (and 3-byte addresses) transferred most significant bit/byte first.

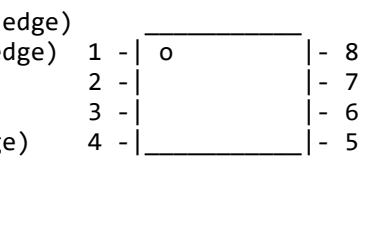
DSi "5A32" chip (32Kit aka 4Kbyte)

Early DSi DWM-W015 boards did have 128Kbyte FLASH, but later boards have custum 4Kbyte FLASH chips (these 4K chips are found on later DSi DWM-W015 boards, and DSi DWM-W024 boards, and 3DS DWM-W028 boards). The chips are having a 24bit address bus (needed for NDS compatibility), and, a weird non-writeable gap within a 128Kbyte memory are:

```
  000000h..0002FFh  Writeable only if /WP=HIGH (otherwise writes are ignored)
  000300h..01F2FFh  Not writeable (FFh-filled, writes are ignored)
```

```
  01F300h..01FFFFh  Writeable
  020000h and up    Mirrors of 0..01FFFFh (same read/write-ability as above)
```
There are several part numbers: "5A32" (DSi), "5K32" (3DS), "32B, 3XH" (3DS), and "26FV032T" (DSi), that chips are probably all the same size & functionally same; most of those 4Kbyte chips have tiny packages (except "26FV032T" which comes in classic large package).

Pin-Outs (Large Package, in NDS, and early DSi boards)
```
  1   D    Serial Data In (latched at rising clock edge)
  2   C    Serial Clock (max 25MHz)
  3   /RES Reset
  4   /S   Chip Select (instructions start at falling edge)
  5   /W   Write Protect (makes first 256 pages read-only)
  6   VCC  Supply (2.7V..3.6V typ) (4V max) (DS:VDD3.3)
  7   VSS  Ground
  8   Q    Serial Data Out (changes at falling clock edge)
```


Pin-Outs (Tiny Package, in newer DSi boards, and 3DS)
```
  1   /S   Chip Select (instructions start at falling edge)
  2   Q    Serial Data Out (changes at falling clock edge)
  3   /W   Write Protect (makes first pages read-only)
  4   VSS  Ground
  5   D    Serial Data In (latched at rising clock edge)
  6   C    Serial Clock
  7   /RES Reset
  8   VCC  Supply (2.7V..3.6V typ) (DSi: VDD33)
```


DS Firmware Header

Firmware Memory Map
```
  00000h-00029h  Firmware Header
  0002Ah-001FFh  Wifi Settings
  00200h-3F9FFh  Firmware Code/Data                      ;-NDS only (not DSi)
  00200h-002FEh  00h-filled                              ;\
  002FFh         80h                                     ;
  00300h-1F2FFh  FFh-filled (not write-able on 4K chips)  ; DSi only (not NDS)
  1F300h-1F3FEh  FFh-filled (write-able)                 ;
  1F3FFh         Whatever Debug/Bootflags                ;
  1F400h-1F5FFh  Wifi Access Point 4  ;\with WPA/WPA2     ;
  1F600h-1F7FFh  Wifi Access Point 5  ; support          ;
  1F800h-1F9FFh  Wifi Access Point 6  ;/                 ;/
  3FA00h-3FAFFh  Wifi Access Point 1  ;\
  3FB00h-3FBFFh  Wifi Access Point 2  ; Open/WEP only
  3FC00h-3FCFFh  Wifi Access Point 3  ;/
  3FD00h-3FDFFh  Not used
  3FE00h-3FEFFh  User Settings Area 1
  3FF00h-3FFFFh  User Settings Area 2
```
On iQue DS (with 512K flash memory), user settings are moved to 7FE00h and up, and, there seems to be some unknown stuff at 200h..27Fh.

Firmware Header (00000h-001FFh)
```
  Addr Size Expl.
  000h 2    part3 romaddr/8 (arm9 gui code) (LZ/huffman compression)
  002h 2    part4 romaddr/8 (arm7 wifi code) (LZ/huffman compression)
  004h 2    part3/4 CRC16 arm9/7 gui/wifi code
  006h 2    part1/2 CRC16 arm9/7 boot code
  008h 4    firmware identifier (usually nintendo "MAC",nn) (or nocash "XBOO")
            the 4th byte (nn) occassionally changes in different versions
  00Ch 2    part1 arm9 boot code romaddr/2^(2+shift1) (LZSS compressed)
  00Eh 2    part1 arm9 boot code 2800000h-ramaddr/2^(2+shift2)
  010h 2    part2 arm7 boot code romaddr/2^(2+shift3) (LZSS compressed)
```

```
  012h 2    part2 arm7 boot code 3810000h-ramaddr/2^(2+shift4)
  014h 2    shift amounts, bit0-2=shift1, bit3-5=shift2, bit6-8=shift3,
            bit9-11=shift4, bit12-15=firmware_chipsize/128K
  016h 2    part5 data/gfx romaddr/8 (LZ/huffman compression)
  018h 8    Optional KEY1-encrypted "enPngOFF"=Cartridge KEY2 Disable
            (feature isn't used in any consoles, instead contains timestamp)
  018h 5    Firmware version built timestamp (BCD minute,hour,day,month,year)
  01Dh 1    Console type
              FFh=Nintendo DS
              20h=Nintendo DS-lite
              57h=Nintendo DSi (also iQueDSi)
              43h=iQueDS
              63h=iQueDS-lite
            The entry was unused (FFh) in older NDS, ie. replace FFh by 00h)
              Bit0   seems to be DSi/iQue related
              Bit1   seems to be DSi/iQue related
              Bit2   seems to be DSi related
              Bit3   zero
              Bit4   seems to be DSi related
              Bit5   seems to be DS-Lite related
              Bit6   indicates presence of "extended" user settings (DSi/iQue)
              Bit7   zero
  01Eh 2    Unused (FFh-filled)
  020h 2    User Settings Offset (div8) (usually last 200h flash bytes)
  022h 2    Unknown (7EC0h or 0B51h)
  024h 2    Unknown (7E40h or 0DB3h)
  026h 2    part5 CRC16 data/gfx
  028h 2    unused (FFh-filled)
  02Ah-1FFh Wifi Calibration Data (see next chapter)
```

DSi

```
  000h 1Dh   Zerofilled (bootcode is in new eMMC chip, not on old FLASH chip)
  01Dh 6     Same as on DS (header: Console Type and User Settings Offset)
  022h 6     Zerofilled (bootcode is in new eMMC chip, not on old FLASH chip)
  028h..1FCh Same as on DS (wifi calibration)
  1FDh 1     Wifi Board (01h=DWM-W015, 02h=W024, 03h=W028)   ;\this was
  1FEh 1     Wifi Flash (20h=With access point 4/5/6)        ; FFh-filled
  1FFh 1     Same as on DS (FFh)                             ;/on DS
  200h FFh   Zerofilled                                      ;\
  2FFh 1     Unknown (80h)                                   ; this was
  300h   1F000h FFh's (not write-able on 4K chips)           ; bootcode
  1F300h FFh    FFh's (write-able) ;twl-debugger: 00h's      ; on DS
  1F3FFh 1      FFh                ;twl-debugger: 40h        ;/
```

The bytes [000h..027h] cannot be changed on DSi because they are part of the RSA signature in DSi's Boot Info Block (at eMMC offset 200h..3FFh).

DS Firmware Wifi Calibration Data

Wifi Calibration/Settings (located directly after Firmware Header)

```
  Addr Size Expl.
  000h-029h Firmware Header (see previous chapter)
  02Ah 2    CRC16 (with initial value 0) of [2Ch..2Ch+config_length-1]
  02Ch 2    config_length (usually 0138h, ie. entries 2Ch..163h)
  02Eh 1    Unused        (00h)
  02Fh 1    Version (0=v1..v4, 3=v5, 5=v6..v7,6=W006,15=W015,24=W024,34=N3DS)
  030h 6    Unused        (00h-filled) (DS-Lite and DSi: FF,FF,FF,FF,FF,00)
  036h 6    48bit MAC address (v1-v5: 0009BFxxxxxx, v6-v7: 001656xxxxxx)
  03Ch 2    list of enabled channels ANDed with 7FFE (Bit1..14 = Channel 1..14)
            (usually 3FFEh, ie. only channel 1..13 enabled)
  03Eh 2    Whatever Flags (usually FFFFh)
  040h 1    RF Chip Type (NDS: usually 02h) (DS-Lite and DSi/3DS: usually 03h)
```

```
  041h 1    RF Bits per entry at 0CEh (usually 18h=24bit=3byte) (Bit7=?)
  042h 1    RF Number of entries at 0CEh (usually 0Ch)
  043h 1    Unknown (usually 01h)
  044h 2    Initial Value for [4808146h]  ;W_CONFIG_146h
  046h 2    Initial Value for [4808148h]  ;W_CONFIG_148h
  048h 2    Initial Value for [480814Ah]  ;W_CONFIG_14Ah
  04Ah 2    Initial Value for [480814Ch]  ;W_CONFIG_14Ch
  04Ch 2    Initial Value for [4808120h]  ;W_CONFIG_120h
  04Eh 2    Initial Value for [4808122h]  ;W_CONFIG_122h
  050h 2    Initial Value for [4808154h]  ;W_CONFIG_154h
  052h 2    Initial Value for [4808144h]  ;W_CONFIG_144h
  054h 2    Initial Value for [4808130h]  ;W_CONFIG_130h
  056h 2    Initial Value for [4808132h]  ;W_CONFIG_132h
  058h 2    Initial Value for [4808140h]  ;W_CONFIG_140h
  05Ah 2    Initial Value for [4808142h]  ;W_CONFIG_142h
  05Ch 2    Initial Value for [4808038h]  ;W_POWER_TX
  05Eh 2    Initial Value for [4808124h]  ;W_CONFIG_124h
  060h 2    Initial Value for [4808128h]  ;W_CONFIG_128h
  062h 2    Initial Value for [4808150h]  ;W_CONFIG_150h
  064h 69h  Initial 8bit values for BB[0..68h]
  0CDh 1    Unused (00h)
```

Below for Type2 (ie. when [040h]=2) (Mitsumi MM3155 and RF9008):

```
  0CEh 24h  Initial 24bit values for RF[0,4,5,6,7,8,9,0Ah,0Bh,1,2,3]
  0F2h 54h  Channel 1..14 2x24bit values for RF[5,6]
  146h 0Eh  Channel 1..14 8bit values for BB[1Eh] (usually somewhat B1h..B7h)
  154h 0Eh  Channel 1..14 8bit values for RF[9].Bit10..14 (usually 10h-filled)
```

Below for Type3 (ie. when [040h]=3) (Mitsumi MM3218) (and AR6013G):

```
  --- Type3 values are originated at 0CEh, following addresses depend on:  ---
  1) number of initial values, found at [042h]        ;usually 29h
  2) number of BB indices,     found at [0CEh+[042h]] ;usually 02h
  3) number of RF indices,     found at [043h]        ;usually 02h
  --- Below example addresses assume above values to be set to 29h,02h,02h ---
  0CEh 29h  Initial 8bit values for RF[0..28h]
  0F7h 1    Number of BB indices per channel
  0F8h 1    1st BB index
  0F9h 14   1st BB data for channel 1..14
  107h 1    2nd BB index
  108h 14   2nd BB data for channel 1..14
  116h 1    1st RF index
  117h 14   1st RF data for channel 1..14
  125h 1    2nd RF index
  126h 14   2nd RF data for channel 1..14
  134h 46   Unused (FFh-filled)
```

Below for both Type2 and Type3:

```
  162h 1    Unknown (usually 19h..1Ch)
  163h 1    Unused (FFh) (Inside CRC16 region, with config_length=138h)
  164h 99h  Unused (FFh-filled) (Outside CRC16 region, with config_length=138h)
  1FDh 1    DSi/3DS Wifi Board (01h=W015, 02h=W024, 03h=W028);\this was
  1FEh 1    DSi/3DS Wifi Flash (20h=With access point 4/5/6) ; FFh-filled on DS
  1FFh 1    DSi/3DS Same as on DS (FFh)                      ;/
```

Most of the Wifi settings seem to be always the same values on all currently existing consoles. Except for:
Values that are (obviously) different are the CRC16, and 4th-6th bytes of the MAC address. Also, initial values for BB[01h] and BB[1Eh], and channel 1..14 values for BB[1Eh], and unknown entry [162h] contain different calibration settings on all consoles.
Firmware v5 is having a new wifi ID [2Fh]=03h, and different RF[9] setting.
Firmware v6 (dslite) has wifi ID [2Fh]=05h, and same RF[9] setting as v5, additionally, v6 and up have different 2nd-3rd bytes of the MAC address.

Moreover, a LOT of values are different with Type3 chips (ie. when [040h]=3).

Note
Unlike for Firmware User Settings, the Firmware Header (and Wifi Settings) aren't stored in RAM upon boot.

So the data must be retrieved via SPI bus by software.

DS Firmware Wifi Internet Access Points

Connection data 1 at WifiFlash[00020h]*8-400h (eg. 01FA00h/03FA00h/07FA00h)
Connection data 2 at WifiFlash[00020h]*8-300h (eg. 01FB00h/03FB00h/07FB00h)
Connection data 3 at WifiFlash[00020h]*8-200h (eg. 01FC00h/03FC00h/07FC00h)
These three 100h byte regions are used to memorize known internet access points. The NDS firmware doesn't use these regions, but games that support internet are allowed to read (and configure/write) them. The DSi firmware also supports configuring these entries.

```
  Addr Siz Expl.
  000h 64  Unknown (usually 00h-filled) (no Proxy supported on NDS)
  040h 32  SSID (ASCII name of the access point) (padded with 00h's)
  060h 32  SSID for WEP64 on AOSS router (each security level has its own SSID)
  080h 16  WEP Key 1 (for type/size, see entry E6h)
  090h 16  WEP Key 2  ;\
  0A0h 16  WEP Key 3  ; (usually 00h-filled)
  0B0h 16  WEP Key 4  ;/
  0C0h 4   IP Address           (0=Auto/DHCP)
  0C4h 4   Gateway              (0=Auto/DHCP)
  0C8h 4   Primary DNS Server   (0=Auto/DHCP)
  0CCh 4   Secondary DNS Server (0=Auto/DHCP)
  0D0h 1   Subnet Mask (0=Auto/DHCP, 1..1Ch=Leading Ones) (eg. 6 = FC.00.00.00)
  0D1h ..  Unknown (usually 00h-filled)
  0E6h 1   WEP Mode (0=None, 1/2/3=5/13/16 byte hex, 5/6/7=5/13/16 byte ascii)
  0E7h 1   Status (00h=Normal, 01h=AOSS, FFh=connection not configured/deleted)
  0E8h 1   Zero (not SSID Length, ie. unlike as entry 4,5,6 on DSi)
  0E9h 1   Unknown (usually 00h)
  0EAh 2   DSi only: MTU (Max transmission unit) (576..1500, usually 1400)
  0ECh 3   Unknown (usually 00h-filled)
  0EFh 1   bit0/1/2 - connection 1/2/3 (1=Configured, 0=Not configured)
  0F0h 6   Nintendo Wifi Connection (WFC) 43bit User ID
           (ID=([F0h] AND 07FFFFFFFFFFh)*1000, shown as decimal string
           NNNN-NNNN-NNNN-N000) (the upper 5bit of the last byte are
           containing additional/unknown nonzero data)
  0F6h 8   Unknown (nonzero stuff !?!)
  0FEh 2   CRC16 for Entries 000h..0FDh (with initial value 0000h)
```

For connection 3: entries [0EFh..0FDh] - always zero-filled?
The location of the first data block is at the User Settings address minus 400h, ie. Firmware Header [00020h]*8-400h.

Connection data 4 at WifiFlash[00020h]*8-A00h (eg. 01F400h) (DSi only)
Connection data 5 at WifiFlash[00020h]*8-800h (eg. 01F600h) (DSi only)
Connection data 6 at WifiFlash[00020h]*8-600h (eg. 01F800h) (DSi only)
The DSi has three extra 200h-byte regions (for use DSi games, with the new WPA/WPA2 encryption support, and with additional proxy support), these extra regions are found under "Advanced Setup" in the DSi firmware's "Internet" configuration menu.

```
  Addr Siz Expl.
  000h 32  Proxy Authentication Username (ASCII string, padded with 00's)
  000h 32  Proxy Authentication Password (ASCII string, padded with 00's)
  040h 32  SSID (ASCII string, padded with 00's) (see [0E8h] for length)
  060h ..  Maybe same as NDS
  080h 16  WEP Key (zerofilled for WPA)
  0xxh ..  Maybe same as NDS
  0C0h 4   IP Address           (0=Auto/DHCP)
  0C4h 4   Gateway              (0=Auto/DHCP)
  0C8h 4   Primary DNS Server   (0=Auto/DHCP)
  0CCh 4   Secondary DNS Server (0=Auto/DHCP)
  0D0h 1   Subnet Mask (0=Auto/DHCP, 1..1Ch=Leading Ones) (eg. 6 = FC.00.00.00)
```

```
  0D1h ..  Unknown (zerofilled)
  0E6h 1   WEP (00h=None/WPA/WPA2, 01h/02h/03h/05h/06h/07h=WEP, same as NDS)
  0E7h 1   WPA (00h=Normal, 10h=WPA/WPA2, 13h=WPS+WPA/WPA2, FFh=unused/deleted)
  0E8h 1   SSID Length in characters (01h..20h, or 00h=unused)
  0E9h 1   Unknown (usually 00h)
  0EAh 2   MTU Value (Max transmission unit) (576..1500, usually 1400)
  0ECh 3   Unknown (usually 00h-filled)
  0EFh 1   bit0/1/2 - connection 4/5/6 (1=Configured, 0=Not configured)
  0F0h 14  Zerofilled (or maybe ID as on NDS, if any such ID exists for DSi?)
  0FEh 2   CRC16 for Entries 000h..0FDh (with initial value 0000h)
  100h 32  Precomputed PSK (based on WPA/WPA2 password and SSID) ;\all zero
  120h 64  WPA/WPA2 password (ASCII string, padded with 00's)    ;/for WEP
  160h 33  Zerofilled
  181h 1   WPA (0=None/WEP, 4=WPA-TKIP, 5=WPA2-TKIP, 6=WPA-AES, 7=WPA2-AES)
  182h 1   Proxy Enable         (00h=None, 01h=Yes)
  183h 1   Proxy Authentication (00h=None, 01h=Yes)
  184h 48  Proxy Name (ASCII string, max 47 chars, padded with 00's)
  1B4h 52  Zerofilled
  1E8h 2   Proxy Port (16bit)
  1EAh 20  Zerofilled
  1FEh 2   CRC16 for Entries 100h..1FDh (with initial value 0000h) (0=deleted)
```

The location of the first data block (aka settings number 4) is at the User Settings address minus A00h, ie. Firmware Header [00020h]*8-A00h.

Observe that NDS consoles do have NDS Firmware bootcode/data in that area, so those new regions can exist on DSi only (or on homebrew NDS firmwares). Presence of the new regions is indicated in Firmware Header [001FEh], that byte is usually FFh=NDS or 20h=DSi, the DSi browser does internally replace FFh by 10h, and does then check if byte>=20h (ie. the new areas exist if the byte is 20h..FEh).

Note that the Proxy feature can be used to redirect internet access (when using a custom proxy server, one could redirect commercial games to homebrew servers; as done by the http://pbsds.net/ project) (actually the same should be possible with the DNS server entry, possibly with less traffic).

Note

The location of the user settings & connection data varies (eg. 01Fxxxh=DSi, 03Fxxxh=NDS, 07Fxxxh=iQueDS).

DS Firmware User Settings

Current Settings (RAM 27FFC80h-27FFCEFh)
User Settings 0 (Firmware 3FE00h-3FEFFh) ;(DSi & iQue use different address,
User Settings 1 (Firmware 3FF00h-3FFFFh) ;see Firmware Header [020h])

```
  Addr Size Expl.
  000h  2   Version (5) (Always 5, for all NDS/DSi Firmware versions)
  002h  1   Favorite color (0..15) (0=Gray, 1=Brown, etc.)
  003h  1   Birthday month (1..12) (Binary, non-BCD)
  004h  1   Birthday day   (1..31) (Binary, non-BCD)
  005h  1   Not used (zero)
  006h  20  Nickname string in UTF-16 format
  01Ah  2   Nickname length in characters    (0..10)
  01Ch  52  Message string in UTF-16 format
  050h  2   Message length in characters     (0..26)
  052h  1   Alarm hour     (0..23) (Binary, non-BCD)
  053h  1   Alarm minute   (0..59) (Binary, non-BCD)
  054h  2
  056h  1   80h=enable alarm (huh?), bit 0..6=enable?
  057h  1   Zero (1 byte)
  058h  2x2 Touch-screen calibration point (adc.x1,y1) 12bit ADC-position
  05Ch  2x1 Touch-screen calibration point (scr.x1,y1) 8bit pixel-position
  05Eh  2x2 Touch-screen calibration point (adc.x2,y2) 12bit ADC-position
  062h  2x1 Touch-screen calibration point (scr.x2,y2) 8bit pixel-position
```

```
  064h  2   Language and Flags (see below)
  066h  1   Year (2000..2255) (when having entered date in the boot menu)
  067h  1   Unknown (usually 00h...08h or 78h..7Fh or so)
  068h  4   RTC Offset (difference in seconds when RTC time/date was changed)
  06Ch  4   Not used (FFh-filled, sometimes 00h-filled) (=MSBs of above?)
```
Below not stored in RAM (found only in FLASH memory)...
```
  070h  2   update counter (used to check latest) (must be 0000h..007Fh)
  072h  2   CRC16 of entries 00h..6Fh (70h bytes)
  074h  8Ch Not used (FFh-filled) (or extended data, see below)
```
Below extended data was invented for iQue DS (for adding the chinese language setting), and is also included in Nintendo DSi models. Presence of extended data is indicated in Firmware Header entry [1Dh].Bit6.
```
  074h  1   Unknown (01h) (maybe version?)
  075h  1   Extended Language (0..5=Same as Entry 064h, plus 6=Chinese)
            (for language 6, entry 064h defaults to english; for compatibility)
            (for language 0..5, both entries 064h and 075h have same value)
  076h  2   Bitmask for Supported Languages (Bit0..6)
            (007Eh for iQue DS, ie. with chinese, but without japanese)
            (0042h for iQue DSi, chinese (and english, but only for NDS mode))
            (003Eh for DSi/EUR, ie. without chinese, and without japanese)
  078h  86h Not used (FFh-filled on iQue DS, 00h-filled on DSi)
  0FEh  2   CRC16 of entries 74h..FDh (8Ah bytes)
```
Note: The DSi has some more settings (eg. Country (additionally to Language), Parental Controls, and a surreal fake Wireless-Disable option; which does only disable the Wifi LED, the actual Wifi transmission does still work).
That new settings are stored in eMMC files. The old/above User Settings are stored in those files too (and copy of those User Settings is stored in Wifi FLASH, as described above; that copy is intended mainly for backwards compatibilty with NDS games).
DSi SD/MMC Firmware System Settings Data Files
DSi Backlight level and DSi sound volume seem to be stored in the BPTWL chip (or possibly in its attached I2C potentiometer).

Language and Flags (Entry 064h)
```
  Bit
  0..2 Language (0=Japanese, 1=English, 2=French, 3=German,
       4=Italian, 5=Spanish, 6..7=Reserved) (for Chinese see Entry 075h)
       (the language setting also implies time/data format)
  3    GBA mode screen selection (0=Upper, 1=Lower)
  4-5  Backlight Level    (0..3=Low,Med,High,Max) (DS-Lite only)
  6    Bootmenu Disable   (0=Manual/bootmenu, 1=Autostart Cartridge)
  9    Settings Lost (1=Prompt for User Info, and Language, and Calibration)
  10   Settings Okay (0=Prompt for User Info)
  11   Settings Okay (0=Prompt for User Info) (Same as Bit10)
  12   No function
  13   Settings Okay (0=Prompt for User Info, and Language)
  14   Settings Okay (0=Prompt for User Info) (Same as Bit10)
  15   Settings Okay (0=Prompt for User Info) (Same as Bit10)
```
The Health and Safety message is skipped if Bit9=1, or if one or more of the following bits is zero: Bits 10,11,13,14,15. However, as soon as entering the bootmenu, the Penalty-Prompt occurs.

Note: There are two User Settings areas in the firmware chip, at offset 3FE00h and 3FF00h, if both areas have valid CRCs, then the current/newest area is that whose Update Counter is one bigger than in the other/older area.
```
  IF count1=((count0+1) AND 7Fh) THEN area1=newer ELSE area0=newer
```
When changing settings, the older area is overwritten with new data (and incremented Update Counter). The two areas allow to recover previous settings in case of a write-error (eg. on a battery failure during write).

Battery Removal
Even though the battery is required only for the RTC (not for the firmware flash memory), most of the firmware user settings are reset when removing the battery. This appears to be a strange bug-or-feature of the DS bios, at least, fortunately, it still keeps the rest of the firmware intact.

DS Firmware Extended Settings

Extended Settings contain some additional information which is not supported by the original firmware (current century, date/time formats, temperature calibration, etc.), the settings are supported by Nocash Firmware, by the no$gba emulator, and may be eventually also supported by other emulators. If present, the values can be used by games, otherwise games should use either whatever default settings, or contain their own configuration menu.

Extended Settings - loaded to 23FEE00h (aka fragments of NDS9 boot code)

```
  Addr Siz Expl.
  00h  8   ID "XbooInfo"
  08h  2   CRC16 Value [0Ch..0Ch+Length-1]
  0Ah  2   CRC16 Length (from 0Ch and up)
  0Ch  1   Version (currently 01h)
  0Dh  1   Update Count (newer = (older+1) AND FFh)
  0Eh  1   Bootmenu Flags
             Bit6   Important Info  (0=Disable, 1=Enable)
             Bit7   Bootmenu Screen (0=Upper, 1=Lower)
  0Fh  1   GBA Border (0=Black, 1=Gray Line)
  10h  2   Temperature Calibration TP0 ADC value  (x16) (sum of 16 ADC values)
  12h  2   Temperature Calibration TP1 ADC value  (x16) (sum of 16 ADC values)
  14h  2   Temperature Calibration Degrees Kelvin (x100) (0=none)
  16h  1   Temperature Flags
             Bit0-1 Format (0=Celsius, 1=Fahrenheit, 2=Reaumur, 3=Kelvin)
  17h  1   Backlight Intensity (0=0ff .. FFh=Full)
  18h  4   Date Century Offset       (currently 20, for years 2000..2099)
  1Ch  1   Date Month Recovery Value (1..12)
  1Dh  1   Date Day Recovery Value   (1..31)
  1Eh  1   Date Year Recovery Value  (0..99)
  1Fh  1   Date/Time Flags
             Bit0-1 Date Format   (0=YYYY-MM-DD, 1=MM-DD-YYYY, 2=DD-MM-YYYY)
             Bit2   Friendly Date (0=Raw Numeric, 1=With Day/Month Names)
             Bit5   Time DST      (0=Hide DST, 1=Show DST=On/Off)
             Bit6   Time Seconds  (0=Hide Seconds, 1=Show Seconds)
             Bit7   Time Format   (0=24 hour, 1=12 hour)
  20h  1   Date Separator      (Ascii, usually Slash, or Dot)
  21h  1   Time Separator      (Ascii, usually Colon, or Dot)
  22h  1   Decimal Separator   (Ascii, usually Comma, or Dot)
  23h  1   Thousands Separator (Ascii, usually Comma, or Dot)
  24h  1   Daylight Saving Time (Nth)
              Bit 0-3 Activate on (0..4 = Last,1st,2nd,3rd,4th)
              Bit 4-7 Deactivate on (0..4 = Last,1st,2nd,3rd,4th)
  25h  1   Daylight Saving Time (Day)
              Bit 0-3 Activate on (0..7 = Mon,Tue,Wed,Thu,Fri,Sat,Sun,AnyDay)
              Bit 4-7 Deactivate on (0..7 = Mon,Tue,Wed,Thu,Fri,Sat,Sun,AnyDay)
  26h  1   Daylight Saving Time (of Month)
              Bit 0-3 Activate DST in Month   (1..12)
              Bit 4-7 Deactivate DST in Month (1..12)
  27h  1   Daylight Saving Time (Flags)
              Bit 0   Current DST State (0=Off, 1=On)
              Bit 1   Adjust DST Enable (0=Disable, 1=Enable)
```

Note: With the original firmware, the memory region at 23FEE00h and up contains un-initialized, non-zero-filled data (fragments of boot code).

DS Wireless Communications

[DS Wifi I/O Map]
[DS Wifi Control]

2.4GHz band, Wireless LAN (WLAN) IEEE802.11b protocol

Credits
A very large part of the DS Wifi chapters is based on Stephen Stair's great DS Wifi document, thanks there.

DS Wifi I/O Map

Notice
Wifi Registers & RAM cannot be written to by STRB opcodes (ignored).

Registers - NDS7 - 4808000h..4808FFFh

```
  Address  Dir   Name            r/w  [Init] Description
  4808000h R     W_ID            ---- [1440] Chip ID (1440h=DS, C340h=DS-Lite)
  4808004h R/W   W_MODE_RST      9fff [0000] Mode/Reset
  4808006h R/W   W_MODE_WEP      --7f [0000] Mode/Wep modes
  4808008h R/W   W_TXSTATCNT     ffff [0000] Beacon Status Request
  480800Ah R/W   W_X_00Ah        ffff [0000] [bit7 - ingore rx duplicates]
  4808010h R/W   W_IF            ackk [0000] Wifi Interrupt Request Flags
  4808012h R/W   W_IE            ffff [0000] Wifi Interrupt Enable
  4808018h R/W   W_MACADDR_0     ffff [0000] Hardware MAC Address, 1st 2 bytes
```

```
  480801Ah R/W   W_MACADDR_1     ffff [0000] Hardware MAC Address, next 2 bytes
  480801Ch R/W   W_MACADDR_2     ffff [0000] Hardware MAC Address, last 2 bytes
  4808020h R/W   W_BSSID_0       ffff [0000] BSSID (first 2 bytes)
  4808022h R/W   W_BSSID_1       ffff [0000] BSSID (next 2 bytes)
  4808024h R/W   W_BSSID_2       ffff [0000] BSSID (last 2 bytes)
  4808028h R/W   W_AID_LOW       ---f [0000] usually as lower 4bit of AID value
  480802Ah R/W   W_AID_FULL      -7ff [0000] AID value assigned by a BSS.
  480802Ch R/W   W_TX_RETRYLIMIT ffff [0707] Tx Retry Limit (set from 00h-FFh)
  480802Eh R/W   W_INTERNAL      ---1 [0000]
  4808030h R/W   W_RXCNT         ff0e [0000] Receive control
  4808032h R/W   W_WEP_CNT       ffff [0000] WEP engine enable
  4808034h R?    W_INTERNAL      0000 [0000] bit0,1 (see ports 004h,040h,1A0h)
```

Power-Down Registers (and Random Generator)

```
  4808036h R/W   W_POWER_US      ---3 [0001]
  4808038h R/W   W_POWER_TX      ---7 [0003]
  480803Ch R/W   W_POWERSTATE    -r-2 [0200]
  4808040h R/W   W_POWERFORCE    8--1 [0000]
  4808044h R     W_RANDOM        0xxx [0xxx]
  4808048h R/W   W_POWER_?       ---3 [0000]
```

WLAN Memory Ports

```
  4808050h R/W   W_RXBUF_BEGIN   ffff [4000]
  4808052h R/W   W_RXBUF_END     ffff [4800]
  4808054h R     W_RXBUF_WRCSR   0rrr [0000]
  4808056h R/W   W_RXBUF_WR_ADDR -fff [0000]
  4808058h R/W   W_RXBUF_RD_ADDR 1ffe [0000]
  480805Ah R/W   W_RXBUF_READCSR -fff [0000]
  480805Ch R/W   W_RXBUF_COUNT   -fff [0000]
  4808060h R     W_RXBUF_RD_DATA rrrr [xxxx]
  4808062h R/W   W_RXBUF_GAP     1ffe [0000]
  4808064h R/W   W_RXBUF_GAPDISP -fff [0000]
  4808068h R/W   W_TXBUF_WR_ADDR 1ffe [0000]
  480806Ch R/W   W_TXBUF_COUNT   -fff [0000]
  4808070h W     W_TXBUF_WR_DATA xxxx [xxxx]
  4808074h R/W   W_TXBUF_GAP     1ffe [0000]
  4808076h R/W   W_TXBUF_GAPDISP 0fff [0000]
```

xxx

```
  4808078h W     W_INTERNAL      mirr [mirr] Read: Mirror of 068h
  4808080h R/W   W_TXBUF_BEACON  ffff [0000] Beacon Transmit Location
  4808084h R/W   W_TXBUF_TIM     --ff [0000] Beacon TIM Index in Frame Body
  4808088h R/W   W_LISTENCOUNT   --ff [0000] Listen Count
  480808Ch R/W   W_BEACONINT     -3ff [0064] Beacon Interval
  480808Eh R/W   W_LISTENINT     --ff [0000] Listen Interval
  4808090h R/W   W_TXBUF_CMD     ffff [0000]    (used by firmware part4)
  4808094h R/W   W_TXBUF_REPLY1  ffff [0000]    (used by firmware part4)
  4808098h R     W_TXBUF_REPLY2  0000 [0000]    (used by firmware part4)
  480809Ch R/W   W_INTERNAL      ffff [0050] value 4x00h --> preamble+x*12h us?
  48080A0h R/W   W_TXBUF_LOC1    ffff [0000]
  48080A4h R/W   W_TXBUF_LOC2    ffff [0000]
  48080A8h R/W   W_TXBUF_LOC3    ffff [0000]
  48080ACh W     W_TXREQ_RESET   fixx [0050]
  48080AEh W     W_TXREQ_SET     fixx [0050]
  48080B0h R     W_TXREQ_READ    --1f [0010]
  48080B4h W     W_TXBUF_RESET   0000 [0000]    (used by firmware part4)
  48080B6h R     W_TXBUSY        0000 [0000]    (used by firmware part4)
  48080B8h R     W_TXSTAT        0000 [0000]
  48080BAh ?     W_INTERNAL      0000 [0000]
  48080BCh R/W   W_PREAMBLE      ---3 [0001]
  48080C0h R/W x W_CMD_TOTALTIME ffff [0000]    (used by firmware part4)
  48080C4h R/W x W_CMD_REPLYTIME ffff [0000]    (used by firmware part4)
  48080C8h ?     W_INTERNAL      0000 [0000]
  48080D0h R/W   W_RXFILTER      1fff [0401]
  48080D4h R/W   W_CONFIG_0D4h   ---3 [0001]
  48080D8h R/W   W_CONFIG_0D8h   -fff [0004]
```

```
  48080DAh R/W   W_RX_LEN_CROP   ffff [0602]
  48080E0h R/W   W_RXFILTER2     ---f [0008]
```

Wifi Timers

```
  48080E8h R/W   W_US_COUNTCNT   ---1 [0000] Microsecond counter enable
  48080EAh R/W   W_US_COMPARECNT ---1 [0000] Microsecond compare enable
  48080ECh R/W   W_CONFIG_0ECh   3f1f [3F03]
  48080EEh R/W   W_CMD_COUNTCNT  ---1 [0001]
  48080F0h R/W   W_US_COMPARE0   fc-- [FC00] Microsecond compare, bits 0-15
  48080F2h R/W   W_US_COMPARE1   ffff [FFFF] Microsecond compare, bits 16-31
  48080F4h R/W   W_US_COMPARE2   ffff [FFFF] Microsecond compare, bits 32-47
  48080F6h R/W   W_US_COMPARE3   ffff [FFFF] Microsecond compare, bits 48-63
  48080F8h R/W   W_US_COUNT0     ffff [0000] Microsecond counter, bits 0-15
  48080FAh R/W   W_US_COUNT1     ffff [0000] Microsecond counter, bits 16-31
  48080FCh R/W   W_US_COUNT2     ffff [0000] Microsecond counter, bits 32-47
  48080FEh R/W   W_US_COUNT3     ffff [0000] Microsecond counter, bits 48-63
  4808100h ?     W_INTERNAL      0000 [0000]
  4808102h ?     W_INTERNAL      0000 [0000]
  4808104h ?     W_INTERNAL      0000 [0000]
  4808106h ?     W_INTERNAL      0000 [0000]
  480810Ch R/W   W_CONTENTFREE   ffff [0000] ...
  4808110h R/W   W_PRE_BEACON    ffff [0000]
  4808118h R/W   W_CMD_COUNT     ffff [0000]
  480811Ch R/W   W_BEACONCOUNT1  ffff [0000] reloaded with W_BEACONINT
```

Configuration Ports (and some other Registers)

```
  4808120h R/W   W_CONFIG_120h   81ff [0048] init from firmware[04Ch]
  4808122h R/W   W_CONFIG_122h   ffff [4840] init from firmware[04Eh]
  4808124h R/W   W_CONFIG_124h   ffff [0000] init from firmware[05Eh], or 00C8h
  4808126h ?     W_INTERNAL      fixx [ 0080]
  4808128h R/W   W_CONFIG_128h   ffff [0000] init from firmware[060h], or 07D0h
  480812Ah ?     W_INTERNAL      fixx [1000] lower 12bit same as W_CONFIG_128h
  4808130h R/W   W_CONFIG_130h   -fff [0142] init from firmware[054h]
  4808132h R/W   W_CONFIG_132h   8fff [8064] init from firmware[056h]
  4808134h R/W   W_BEACONCOUNT2  ffff [FFFF] ...
  4808140h R/W   W_CONFIG_140h   ffff [0000] init from firmware[058h], or xx
  4808142h R/W   W_CONFIG_142h   ffff [2443] init from firmware[05Ah]
  4808144h R/W   W_CONFIG_144h   --ff [0042] init from firmware[052h]
  4808146h R/W   W_CONFIG_146h   --ff [0016] init from firmware[044h]
  4808148h R/W   W_CONFIG_148h   --ff [0016] init from firmware[046h]
  480814Ah R/W   W_CONFIG_14Ah   --ff [0016] init from firmware[048h]
  480814Ch R/W   W_CONFIG_14Ch   ffff [162C] init from firmware[04Ah]
  4808150h R/W   W_CONFIG_150h   ff3f [0204] init from firmware[062h], or 202h
  4808154h R/W   W_CONFIG_154h   7a7f [0058] init from firmware[050h]
```

Baseband Chip Ports

```
  4808158h W     W_BB_CNT        mirr [00B5] BB Access Start/Direction/Index
  480815Ah W     W_BB_WRITE      ???? [0000] BB Access data byte to write
  480815Ch R     W_BB_READ       00rr [00B5] BB Access data byte read
  480815Eh R     W_BB_BUSY       000r [0000] BB Access Busy flag
  4808160h R/W   W_BB_MODE       41-- [0100] BB Access Mode
  4808168h R/W   W_BB_POWER      8--f [800D] BB Access Powerdown
```

Internal Stuff

```
  480816Ah ?     W_INTERNAL      0000 [0001] (or 0000h?)
  4808170h ?     W_INTERNAL      0000 [0000]
  4808172h ?     W_INTERNAL      0000 [0000]
  4808174h ?     W_INTERNAL      0000 [0000]
  4808176h ?     W_INTERNAL      0000 [0000]
  4808178h W     W_INTERNAL      fixx [0800] Read: mirror of 17Ch
```

RF Chip Ports

```
  480817Ch R/W   W_RF_DATA2      ffff [0800]
  480817Eh R/W   W_RF_DATA1      ffff [C008]
  4808180h R     W_RF_BUSY       000r [0000]
  4808184h R/W   W_RF_CNT        413f [0018]
```

xxx

```
  4808190h R/W   W_INTERNAL      ffff [0000]
  4808194h R/W   W_TX_HDR_CNT    ---7 [0000] used by firmware part4 (0 or 6)
```

```
  4808198h R/W   W_INTERNAL      ---f [0000]
  480819Ch R     W_RF_PINS       fixx [0004]
  48081A0h R/W   W_X_1A0h        -933 [0000] used by firmware part4 (0 or 823h)
  48081A2h R/W   W_X_1A2h        ---3 [0001] used by firmware part4
  48081A4h R/W   W_X_1A4h        ffff [0000] "Rate used when signal test..."
```

Wifi Statistics

```
  48081A8h R     W_RXSTAT_INC_IF rrrr [0000] Stats Increment Flags
  48081AAh R/W   W_RXSTAT_INC_IE ffff [0000] Stats Increment IRQ Enable
  48081ACh R     W_RXSTAT_OVF_IF rrrr [0000] Stats Half-Overflow Flags
  48081AEh R/W   W_RXSTAT_OVF_IE ffff [0000] Stats Half-Overflow IRQ Enable
  48081B0h R/W   W_RXSTAT        --ff [0000]
  48081B2h R/W   W_RXSTAT        ffff [0000] RX_LengthRateErrorCount
  48081B4h R/W   W_RXSTAT        rrff [0000] ... firmware uses also MSB ... ?
  48081B6h R/W   W_RXSTAT        ffff [0000]
  48081B8h R/W   W_RXSTAT        --ff [0000]
  48081BAh R/W   W_RXSTAT        --ff [0000]
  48081BCh R/W   W_RXSTAT        ffff [0000]
  48081BEh R/W   W_RXSTAT        ffff [0000]
  48081C0h R/W   W_TX_ERR_COUNT  --ff [0000] TransmitErrorCount
  48081C4h R     W_RX_COUNT      fixx [0000]
```

[1D0 - 1DE are 15 entries related to multiplayer response errors]

```
  48081D0h R/W   W_CMD_STAT      ff-- [0000]
  48081D2h R/W   W_CMD_STAT      ffff [0000]
  48081D4h R/W   W_CMD_STAT      ffff [0000]
  48081D6h R/W   W_CMD_STAT      ffff [0000]
  48081D8h R/W   W_CMD_STAT      ffff [0000]
  48081DAh R/W   W_CMD_STAT      ffff [0000]
  48081DCh R/W   W_CMD_STAT      ffff [0000]
  48081DEh R/W   W_CMD_STAT      ffff [0000]
```

Internal Diagnostics Registers (usually not used for anything)

```
  48081F0h R/W   W_INTERNAL      ---3 [0000]
  4808204h ?     W_INTERNAL      fixx [0000]
  4808208h ?     W_INTERNAL      fixx [0000]
  480820Ch W     W_INTERNAL      fixx [0050]
  4808210h R     W_TX_SEQNO      fixx [0000]
  4808214h R     W_RF_STATUS     XXXX [0009]    (used by firmware part4)
  480821Ch W     W_IF_SET        fbff [0000] Force Interrupt (set bits in W_IF)
  4808220h R/W   W_INTERNAL      ffff [0000] Bit0-1: Enable/Disable WifiRAM
                                             (locks memory at 4000h-5FFFh)
  4808224h R/W   W_INTERNAL      ---3 [0003]
  4808228h W     W_X_228h        fixx [0000]    (used by firmware part4) (bit3)
  4808230h R/W   W_INTERNAL      --ff [0047]
  4808234h R/W   W_INTERNAL      -eff [0EFF]
  4808238h R/W   W_INTERNAL      ffff [0000] ;rx_seq_no-60h+/-x   ;why that?
                                   ;other day: fixed value, not seq_no related?
  480823Ch ?     W_INTERNAL      fixx [0000] like W_TXSTAT... ONLY for beacons?
  4808244h R/W   W_X_244h        ffff [0000]    (used by firmware part4)
  4808248h R/W   W_INTERNAL      ffff [0000]
  480824Ch R     W_INTERNAL      fixx [0000] ;rx_mac_addr_0
  480824Eh R     W_INTERNAL      fixx [0000] ;rx_mac_addr_1
  4808250h R     W_INTERNAL      fixx [0000] ;rx_mac_addr_2
  4808254h ?     W_CONFIG_254h   fixx [0000] (read: FFFFh=DS, EEEEh=DS-Lite)
  4808258h ?     W_INTERNAL      fixx [0000]
  480825Ch ?     W_INTERNAL      fixx [0000]
  4808260h ?     W_INTERNAL      fixx [ 0FEF]
  4808264h R     W_INTERNAL      fixx [0000] ;rx_addr_1 (usually "rxtx_addr-x")
  4808268h R     W_RXTX_ADDR     fixx [0005] ;rxtx_addr
  4808270h R     W_INTERNAL      fixx [0000] ;rx_addr_2 (usually "rx_addr_1-1")
  4808274h ?     W_INTERNAL      fixx [ 0001]
  4808278h R/W   W_INTERNAL      ffff [000F]
  480827Ch ?     W_INTERNAL      fixx [ 000A]
  4808290h (R/W) W_X_290h        fixx [FFFF] bit 0 = ? (used by firmware part4)
  4808298h W     W_INTERNAL      fixx [0000]
  48082A0h R/W   W_INTERNAL      ffff [0000]
```

```
48082A2h R     W_INTERNAL      XXXX [7FFF] 15bit shift reg (used during tx?)
48082A4h R     W_INTERNAL      fixx [0000] ;rx_rate_1 not ALWAYS same as 2C4h
48082A8h W     W_INTERNAL      fixx [0000]
48082ACh ?     W_INTERNAL      fixx [ 0038]
48082B0h W     W_INTERNAL      fixx [0000]
48082B4h R/W   W_INTERNAL      -1-3 [0000]
48082B8h ?     W_INTERNAL      fixx [0000]
48082C0h R/W   W_INTERNAL      ---1 [0000]
48082C4h R     W_INTERNAL      fixx [000A] ;rx_rate_2 (0Ah,14h = 1,2 Mbit/s)
48082C8h R     W_INTERNAL      fixx [0000] ;rx_duration/length/rate (or so?)
48082CCh R     W_INTERNAL      fixx [0000] ;rx_framecontrol; from ieee header
48082D0h DIS   W_INTERNAL                  ;"W_POWERACK" (internal garbage)
                                           ;normally DISABLED (unless FORCE)
48082F0h R/W   W_INTERNAL      ffff [0000]
48082F2h R/W   W_INTERNAL      ffff [0000]
48082F4h R/W   W_INTERNAL      ffff [0000]
48082F6h R/W   W_INTERNAL      ffff [0000]
```

All other ports in range 4808000h..4808FFFh are unused.
All registers marked as "W_INTERNAL" aren't used by Firmware part4, and are probably unimportant, except for whatever special diagnostics purposes.
Reading from write-only ports (W) often mirrors to data from other ports.

Additionally, there are 69h Baseband Chip Registers (BB), and 0Fh RF Chip Registers (see BB and RF chapters).

For Wifi Power Managment (POWCNT2), for Wifi Waitstates (WIFIWAITCNT), see:
DS Power Control
For the Power LED Blink Feature (conventionally used to indicate Wifi activity) see:
DS Power Management Device

For Wifi Configuration and Calibration data in Firmware Header, see:
DS Cartridges, Encryption, Firmware

Wifi RAM - NDS7 - Memory (4804000h..4805FFFh)

```
4804000h W_MACMEM RX/TX Buffers (2000h bytes) (excluding below specials)
4805F60h Used for something, not included in the rx circular buffer.
4805F80h W_WEPKEY_0 (32 bytes)
4805FA0h W_WEPKEY_1 (32 bytes)
4805FC0h W_WEPKEY_2 (32 bytes)
4805FE0h W_WEPKEY_3 (32 bytes)
```

Unlike all other NDS memory, Wifi RAM is left uninitialized after boot.

5F80h - W_WEPKEY_0 thru W_WEPKEY_3 - Wifi WEP keys (R/W)
These WEP key slots store the WEP keys that are used for encryption for 802.11 keys IDs 0-3.

DS Wifi Control

4808000h - W_ID - Wifi Chip ID (R)

```
0-15   Chip ID (1440h on NDS, C340h on NDS-lite)
```

The NDS-lite is more or less backwards compatible with the original NDS (the W_RXBUF_GAPDISP and W_TXBUF_GAPDISP are different, and most of the garbage effects on unused/mirrored ports are different, too).

4808004h - W_MODE_RST - Wifi Hardware mode / reset (R/W)

```
0     Adjust some ports (0/1=see lists below) (R/W)
      TX Master Enable for LOC1..3 and Beacon  (0=Disable, 1=Enable)
1-12  Unknown (R/W)
```

```
  13    Reset some ports (0=No change, 1=Reset/see list below) (Write-Only)
  14    Reset some ports (0=No change, 1=Reset/see list below) (Write-Only)
  15    Unknown (R/W)
```

4808006h - W_MODE_WEP - Wifi Software mode / Wep mode (R/W)

```
  0-2   Unknown, specify a software mode for wifi operation
        (may be related to hardware but a correlation has not yet been found)
  3-5   WEP Encryption Key Size:
         0=Reserved (acts same as 1)
         1=64bit WEP  (IV=24bit + KEY=40bit)  (aka 3+5 bytes)   ;standard/us
         2=128bit WEP (IV=24bit + KEY=104bit) (aka 3+13 bytes)  ;standard/world
         3=152bit WEP (IV=24bit + KEY=128bit) (aka 3+16 bytes)  ;uncommon
         4=Unknown, mabye 256bit WEP (IV=24bit + KEY=232bit) (aka 3+29 bytes)?
         5=Reserved (acts same as 1)
         6=Reserved (acts same as 1)
         7=Reserved (acts same as 1)
  6     Unknown
  8-15  Always zero
```

4808018h - W_MACADDR_0 - MAC Address (R/W)
480801Ah - W_MACADDR_1 - MAC Address (R/W)
480801Ch - W_MACADDR_2 - MAC Address (R/W)

48bit MAC Address of the console. Should be initialized from firmware[036h]. The hardware receives only packets that are sent to this address (or to group addresses, like FF:FF:FF:FF:FF:FF).

4808020h - W_BSSID_0 - BSSID (R/W)
4808022h - W_BSSID_1 - BSSID (R/W)
4808024h - W_BSSID_2 - BSSID (R/W)

48bit BSSID stored here. Ie. the MAC address of the host, obtained from Beacon frames (on the host itself, that should be just same as W_MACADDR). See W_RXFILTER.

4808028h - W_AID_LOW (R/W)

```
  Bit0-3   Maybe player-number, assuming that HW supports such? (1..15, or 0)
  Bit4-15  Not used
```

Usually set equal to the lower 4bit of the W_AID_FULL value.

480802Ah - W_AID_FULL - Association ID (R/W)

```
  Bit0-10  Association ID (AID) (1..2007, or zero)
  Bit11-15 Not used
```

4808032h - W_WEP_CNT - WEP Engine Enable (R/W)

```
  0-14  Unknown (usually zero)
  15    WEP Engine Enable  (0=Disable, 1=Enable)
```

Normally, bit15 should be always set (but it will only affect WEP-encrypted packets, ie. packets with Frame Control bit14=1 in 802.11 header). When disabled, WEP packets aren't received at all (neither in encrypted nor decrypted form), and sending WEP packets might be also ignored(?).
The firmware contains some code that does toggle the bit off for a moment (apparently to reset something after transfer errors).

4808044h - W_RANDOM - Random Generator (R)

```
  0-10  Random
  11-15 Not used (zero)
```

The random generator is updated at 33.51MHz rate, as such:

```
  X = (X AND 1) XOR (X ROL 1)  ;(rotation within 11bit range)
```

That random sequence goes through 5FDh different values before it restarts.
When reading from the random register, the old latched value is returned to the CPU, and the new current random value is then latched, so reads always return the older value, timed from the previous read.
Occassionally, about once every some thousand reads, the latching appears to occur 4 cycles earlier than

normally, so the value on the next read will be 4 cycles older than expected.
The random register has ACTIVE mirrors.

48080BCh - W_PREAMBLE - Preamble Control (R/W)

```
Bit   Dir  Expl.
0     R/W  Unknown                    (this does NOT affect TX)
1     R/W  Preamble (0=Long, 1=Short) (this does NOT affect TX)
2     W    Preamble (0=Long, 1=Short) (this does affect TX) (only at 2Mbit/s)
3-15  -    Always zero
```

Short preamble works only with 2Mbit/s transfer rate (ie. when set like so in TX hardware header). 1Mbit/s rate always uses long preamble.

```
Type   Carrier Signal  SFD Value     PLCP Header     Data
Long   128bit, 1Mbit   16bit, 1Mbit  48bit, 1Mbit    N bits, 1Mbit or 2Mbit
Short  56bit, 1Mbit    16bit, 1Mbit  48bit, 2Mbit    N bits, 2Mbit
```

Length of the Carrier+SFD+PLCP part is thus 192us (long) or 96us (short).
Note: The Carrier+SFD+PLCP part is sent between IRQ14 and IRQ07 (not between IRQ07 and IRQ01).

Writing "0-then-1" to W_MODE_RST.Bit0 does reset following ports:

```
[4808034h]=0002h ;W_INTERNAL
[480819Ch]=0046h ;W_RF_PINS
[4808214h]=0009h ;W_RF_STATUS
[480827Ch]=0005h ;W_INTERNAL
[48082A2h]=?     ;...unstable?
```

Writing "1-then-0" to W_MODE_RST.Bit0 does reset following ports:

```
[480827Ch]=000Ah ;W_INTERNAL
```

Writing "1" to W_MODE_RST.Bit13 does reset following ports:

```
[4808056h]=0000h ;W_RXBUF_WR_ADDR
[48080C0h]=0000h ;W_CMD_TOTALTIME
[48080C4h]=0000h ;W_CMD_REPLYTIME
[48081A4h]=0000h ;W_X_1A4h
[4808278h]=000Fh ;W_INTERNAL
...Also, following may be affected (results are unstable though)...
[48080AEh]=?     ;or rather the actual port (which it is an mirror of)
[48080BAh]=?     ;W_INTERNAL (occassionally unstable)
[4808204h]=?     ;W_INTERNAL
[480825Ch]=?     ;W_INTERNAL
[4808268h]=?     ;W_RXTX_ADDR
[4808274h]=?     ;W_INTERNAL
```

Writing "1" to W_MODE_RST.Bit14 does reset following ports:

```
[4808006h]=0000h ;W_MODE_WEP
[4808008h]=0000h ;W_TXSTATCNT
[480800Ah]=0000h ;W_X_00Ah
[4808018h]=0000h ;W_MACADDR_0
[480801Ah]=0000h ;W_MACADDR_1
[480801Ch]=0000h ;W_MACADDR_2
[4808020h]=0000h ;W_BSSID_0
[4808022h]=0000h ;W_BSSID_1
[4808024h]=0000h ;W_BSSID_2
[4808028h]=0000h ;W_AID_LOW
[480802Ah]=0000h ;W_AID_FULL
[480802Ch]=0707h ;W_TX_RETRYLIMIT
[480802Eh]=0000h ;W_INTERNAL
[4808050h]=4000h ;W_RXBUF_BEGIN
[4808052h]=4800h ;W_RXBUF_END
[4808084h]=0000h ;W_TXBUF_TIM
[48080BCh]=0001h ;W_PREAMBLE
[48080D0h]=0401h ;W_RXFILTER
[48080D4h]=0001h ;W_CONFIG_0D4h
[48080E0h]=0008h ;W_RXFILTER2
```

```
[48080ECh]=3F03h ;W_CONFIG_0ECh
[4808194h]=0000h ;W_TX_HDR_CNT
[4808198h]=0000h ;W_INTERNAL
[48081A2h]=0001h ;W_X_1A2h
[4808224h]=0003h ;W_INTERNAL
[4808230h]=0047h ;W_INTERNAL
```

DS Wifi Interrupts

4808010h - W_IF - Wifi Interrupt Request Flags (R/W)

```
0   Receive Complete  (packet received and stored in the RX fifo)
1   Transmit Complete (packet is done being transmitted) (no matter if error)
2   Receive Event Increment      (IRQ02, see W_RXSTAT_INC_IE)
3   Transmit Error Increment     (IRQ03, see W_TX_ERR_COUNT)
4   Receive Event Half-Overflow  (IRQ04, see W_RXSTAT_OVF_IE)
5   Transmit Error Half-Overflow (IRQ05, see W_TX_ERR_COUNT.Bit7)
6   Start Receive     (IRQ06, a packet has just started to be received)
7   Start Transmit    (IRQ07, a packet has just started to be transmitted)
8   Txbuf Count Expired  (IRQ08, see W_TXBUF_COUNT)
9   Rxbuf Count Expired  (IRQ09, see W_RXBUF_COUNT)
10  Not used (always zero, even when trying to set it with W_IF_SET)
11  RF Wakeup            (IRQ11, see W_POWERSTATE)
12  Multiplay ...?       (IRQ12, see W_CMD_COUNT)
13  Post-Beacon Timeslot (IRQ13, see W_BEACONCOUNT2)
14  Beacon Timeslot      (IRQ14, see W_BEACONCOUNT1/W_US_COMPARE)
15  Pre-Beacon Timeslot  (IRQ15, see W_BEACONCOUNT1/W_PRE_BEACON)
```

Write a '1' to a bit to clear it. For the Half-Overflow flags that works ONLY if the counter MSBs are zero (ie. one must first read the counters (to reset them), and THEN acknowledge the corresponding W_IF bit).
The Transmit Start/Complete bits (Bit7,1) are set for EACH packet (including beacons, and including retries).

4808012h - W_IE - Wifi Interrupt Enable Flags (R/W)

```
0-15  Enable Flags, same bits as W_IF  (0=Disable, 1=Enable)
```

In W_IE, Bit10 is R/W, but seems to have no function since IRQ10 doesn't exist.

480821Ch - W_IF_SET (W_INTERNAL) - Force Wifi Interrupt Flags (W)

```
0-15  Set corresponding bits in W_IF  (0=No change, 1=Set Bit)
```

Notes: Bit10 cannot be set since no IRQ10 exists. This register does only set IRQ flags, but without performing special actions (such like W_BEACONCOUNT1 and W_BEACONCOUNT2 reloads that occur on real IRQ14's).

Wifi Primary IRQ Flag (IF.Bit24, Port 4000214h)

IF.Bit24 gets set <only> when (W_IF AND W_IE) changes from 0000h to non-zero.
IF.Bit24 can be reset (ack) <even> when (W_IF AND W_IE) is still non-zero.

```
Caution  Caution  Caution  Caution  Caution
That means, when acknowledging IF.Bit24, then NO FURTHER wifi IRQs
will be executed whilst and as long as (W_IF AND W_IE) is non-zero.
```

One work-around is to process/acknowledge ALL wifi IRQs in a loop, including further IRQs that may occur inside of that loop, until (W_IF AND W_IE) becomes 0000h.
Another work-around (for single IRQs) would be to acknowledge IF and W_IF, and then to set W_IE temporarily to 0000h, and then back to the old W_IE setting.

DS Wifi Power-Down Registers

4808036h - W_POWER_US (R/W)

```
0     Disable W_US_COUNT and W_BB_ports  (0=Enable, 1=Disable)
1     Unknown (usually 0)
```

```
  2-15  Always zero
```
Bit0=0 enables RFU by setting RFU.Pin11=HIGH, which activates the 22.000MHz oscillator on the RFU board, the 22MHz clock is then output to RFU.Pin26.

4808038h - W_POWER_TX (R/W)
transmit-related power save or sth
init from firmware[05Ch]
```
  0     Auto Wakeup (1=Leave Idle Mode a while after IRQ15)
  1     Auto Sleep  (0=Enter Idle Mode on IRQ13)
  2     Unknown
  3     Unknown (Write-only) (used by firmware)
  4-15  Always zero
```

480803Ch - W_POWERSTATE (R/W)/(R)
```
  0     Unknown (usually 0)                          (R/W)
  1     Request Power Enable (0=No, 1=Yes/queued)    (R/W, but not always)
  2-7   Always zero
  8     Indicates that Bit9 is about the be cleared (Read only)
  9     Current power state (0=Enabled, 1=Disabled) (Read only)
  10-15 Always zero
```
[value =1: queue disable power state] ;<-- seems to be incorrect
[value =2: queue enable power state] ;<-- seems to be correct
Enabling causes wakeup interrupt (IRQ11).
Note: That queue stuff seems to work only if W_POWER_US=0 and W_MODE_RST=1.

4808040h - W_POWERFORCE - Force Power State (R/W)
```
  0     New value for W_POWERSTATE.Bit9  (0=Clear/Delayed, 1=Set/Immediately)
  1-14  Always zero
  15    Apply Bit0 to W_POWERSTATE.Bit9  (0=No, 1=Yes)
```
Setting W_POWERFORCE=8001h whilst W_POWERSTATE.Bit9=0 acts immediately:
```
  (Doing this is okay. Switches to power down mode. Similar to IRQ13.)
  [4808034h]=0002h ;W_INTERNAL
  [480803Ch]=02xxh ;W_POWERSTATE
  [48080B0h]=0000h ;W_TXREQ_READ
  [480819Ch]=0046h ;W_RF_PINS
  [4808214h]=0009h ;W_RF_STATUS (idle)
```
Setting W_POWERFORCE=8000h whilst W_POWERSTATE.Bit9=1 acts delayed:
```
  (Don't do this. After that sequence, the hardware seems to be messed up)
  W_POWERSTATE.Bit8 gets set to indicate the pending operation,
  while pending, changes to W_POWERFORCE aren't applied to W_POWERSTATE,
  while pending, W_POWERACK becomes Read/Write-able,
  writing 0000h to W_POWERACK does clear W_POWERSTATE.Bit8,
  and does apply POWERFORCE.Bit0 to W_POWERSTATE.Bit9
  and does deactivate Port W_POWERACK again.
```

4808048h - W_POWER_? (R/W)
```
  0     Unknown
  1     Unknown
  2-15  Always zero
```
At whatever time (during transmit or so) it gets set to 0003h by hardware.

See also: POWCNT2, W_BB_POWER.

DS Wifi Receive Control

4808030h - W_RXCNT - Wifi Receive Control (parts R/W and W)
```
  0     Copy W_RXBUF_WR_ADDR to W_RXBUF_WRCSR                    (W)
  1-3   Unknown                                                  (R/W)
```

```
  4-6   Always zero
  7     Copy W_TXBUF_REPLY1 to W_TXBUF_REPLY2, set W_TXBUF_REPLY1 to 0000h  (W)
  8-14  Unknown                                                            (R/W)
  15    Enable Queuing received data to RX FIFO                            (R/W)
```

48080D0h - W_RXFILTER - (R/W)

```
  0     (0=Insist on W_BSSID, 1=Accept no matter of W_BSSID)
  1-6   Unknown (usually zero)
  7     Unknown (0 or 1)
  8     Unknown (0 or 1)
  9     Unknown (0 or 1)
  10    Unknown (0 or 1)       (when set, receives beacons, and maybe others)
  11    Unknown (usually zero)     ;reportedly "allow toDS" ?
  12    (0=Normal, 1=Accept even whatever garbage)
  13-15 Not used (always zero)
```

Specifies what packets to allow.
0000h = Disable receive.
FFFFh = Enable receive.
0400h = Receives managment frames (and possibly others, too)

48080E0h - W_RXFILTER2 - (R/W)

```
  0     Unknown (0=Receive Data Frames, 1=Ignore Data Frames) (?)
  1     Unknown
  2     Unknown
  3     Unknown (usually set)
  4-15  Not used (always zero)
```

Firmware writes values 08h, 0Bh, 0Dh (aka 1000b, 1011b, 1101b).
Firmware usually has bit0 set, even when receiving data frames, so, in some situations data frames seem to pass-through even when bit0 is set...? Possibly that situation is when W_BSSID matches...?
Control/PS-Poll frames seem to be passed always (even if W_RXFILTER2=0Fh).

DS Wifi Receive Buffer

The dimensions of the circular Buffer are set with BEGIN/END values, hardware automatically wraps to BEGIN when an incremented pointer hits END address.

Write Area
Memory between WRCSR and READCSR is free for receiving data, the hardware writes incoming packets to this region (to WRCSR and up) (but without exceeding READCSR), once when it has successfully received a complete packet, the hardware moves WRCSR after the packet (aligned to a 4-byte boundary).

Read Area
Memory between READCSR and WRCSR contains received data, which can be read by the CPU via RD_ADDR and RD_DATA registers (or directly from memory). Once when having processed that data, the CPU must set READCSR to the end of it.

4808050h - W_RXBUF_BEGIN - Wifi RX Fifo start location (R/W)
4808052h - W_RXBUF_END - Wifi RX Fifo end location (R/W)

```
  0-15  Byte-offset in Wifi Memory (usually 4000h..5FFEh)
```

Although the full 16bit are R/W, only the 12bit halfword offset in Bit1-12 is actually used, the other bits seem to have no effect.
Some or all (?) of the below incrementing registers are automatically matched to begin/end, that is, after incrementing, IF adr=end THEN adr=begin.

4808054h - W_RXBUF_WRCSR - Wifi RX Fifo Write or "end" cursor (R)

```
  0-11  Halfword Address in RAM
```

```
  12-15 Always zero
```
This is a hardware controlled write location - it shows where the next packet will be written..

4808056h - W_RXBUF_WR_ADDR - Wifi RX Fifo Write Cursor Latch value (R/W)
```
  0-11  Halfword Address in RAM
  12-15 Always zero
```
This is a value that is latched into W_RXBUF_WRCSR, when the W_RXCNT latch bit (W_RXCNT.Bit0) is written.

4808058h - W_RXBUF_RD_ADDR - Wifi CircBuf Read Address (R/W)
```
  0     Always zero
  1-12  Halfword Address in RAM for reading via W_RXBUF_RD_DATA
  13-15 Always zero
```
The circular buffer limits are the same as the range specified for the receive FIFO, however the address can be set outside of that range and will only be affected by the FIFO boundary if it crosses the FIFO end location by reading from the circular buffer.

480805Ah - W_RXBUF_READCSR - Wifi RX Fifo Read or "start" cursor (R/W)
```
  0-11  Halfword Address in RAM
  12-15 Always zero
```
This value is specified the same as W_RXBUF_WRCSR - it's purely software controlled so it's up to the programmer to move the start cursor after loading a packet. If W_RXBUF_READCSR != W_RXBUF_WRCSR, then one or more packets exist in the FIFO that need to be processed (see the section on HW RX Headers, for information on calculating packet lengths). Once a packet has been processed, the software should advance the read cursor to the beginning of the next packet.

4808060h - W_RXBUF_RD_DATA - Wifi CircBuf Read Data (R)
```
  0-15  Data
```
returns the 16bit value at the address specified by W_RXBUF_RD_ADDR, and increments W_RXBUF_RD_ADDR by 2. If the increment causes W_RXBUF_RD_ADDR to equal the address specified in W_RXBUF_END, W_RXBUF_RD_ADDR will be reset to the address specified in W_RXBUF_BEGIN.
Ports 1060h, 6060h, 7060h are PASSIVE mirrors of 0060h, reading from these mirrors returns the old latched value from previous read from 0060h, but without reading a new value from RAM, and without incrementing the address.

4808062h - W_RXBUF_GAP - Wifi RX Gap Address (R/W)
```
  0     Always zero
  1-12  Halfword Address in RAM
  13-15 Always zero
```
Seems to be intended to define a "gap" in the circular buffer, done like so:
```
  Addr=Addr+2 and 1FFEh  ;address increment (by W_RXBUF_RD_DATA read)
  if Addr=RXBUF_END then ;normal begin/end wrapping (done before gap wraps)
     Addr=RXBUF_BEGIN
  if Addr=RXBUF_GAP then ;now gap-wrap (may include further begin/end wrap)
     Addr=RXBUF_GAP+RXBUF_GAPDISP*2
     if Addr>=RXBUF_END then Addr=Addr+RXBUF_BEGIN-RXBUF_END  ;wrap more
```
To disable the gap stuff, set both W_RXBUF_GAP and W_RXBUF_GAPDISP to zero.

4808064h - W_RXBUF_GAPDISP - Wifi RX Gap Displacement Offset (R/W)
```
  0-11  Halfword Offset, used with W_RXBUF_GAP (see there)
  12-15 Always zero
```
Caution: On the DS-Lite, after adding it to W_RXBUF_RD_ADDR, the W_RXBUF_GAPDISP setting is destroyed (reset to 0000h) by hardware. The original DS leaves W_RXBUF_GAPDISP intact.

480805Ch - W_RXBUF_COUNT (R/W)
```
  0-11  Decremented on reads from W_RXBUF_RD_DATA
  12-15 Always zero
```

Triggers IRQ09 when it reaches zero, and does then stay at zero (without further decrementing, and without generating further IRQs).
Note: Also decremented on (accidental) writes to read-only W_RXBUF_RD_DATA.

DS Wifi Receive Statistics

48081A8h - W_RXSTAT_INC_IF - Statistics Increment Flags (R)

```
0-12   Increment Flags (see Port 48081B0h..1BFh)
13-15  Always zero
```

Bitmask for which statistics have been increased at least once.
Unknown how to reset/acknowledge these bits... possibly by reading from 48081A8h, or by reading from 48081B0h..1BFh, or eventually/obscurely by writing to 48081ACh.

48081AAh - W_RXSTAT_INC_IE - Statistics Increment Interrupt Enable (R/W)

```
0-12   Counter Increment Interrupt Enable (see 48081B0h..1BFh) (1=Enable)
13-15  Unknown (usually zero)
```

Statistic Interrupt Enable Control register for Count Up.
Note: ------> seems to trigger IRQ02 ...?

48081ACh - W_RXSTAT_OVF_IF - Statistics Half-Overflow Flags (R)

```
0-12   Half-Overflow Flags (see Port 48081B0h..1BFh)
13-15  Always zero
```

The W_RXSTAT_OVF_IF bits are simply containing the current bit7-value of the corresponding counters, setting or clearing that counter bits is directly reflected to W_RXSTAT_OVF_IF.
The recommended way to acknowledge W_RXSTAT_OVF_IF is to read the corresponding counters (which are reset to 00h after reading). For some reason, the firmware is additionally writing FFFFh to W_RXSTAT_OVF_IF (that is possibly a bug, or it does acknowledge something internally?).

48081AEh - W_RXSTAT_OVF_IE - Statistics Half-Overflow Interrupt Enable (R/W)

```
0-12   Half-Overflow Interrupt Enable (see Port 48081B0h..1BFh) (1=Enable)
13-15  Unknown (usually zero)
```

Statistic Interrupt Enable for Overflow, bits same as in W_RXSTAT_INC_IE
Note: ------> seems to trigger IRQ04 ...?

48081B0h..1BFh - W_RXSTAT - Receive Statistics (R/W, except 1B5h: Read-only)

W_RXSTAT is a collection of 8bit counters, which are incremented upon certain events. These entries are automatically reset to 0000h after reading. Should be accessed with LDRH opcodes (using LDRB to read only 8bits does work, but the read is internally expanded to 16bit, and so, the whole 16bit value will be reset to 0000h).

```
Port      Dir  Bit  Expl.
48081B0h  R/W  0    W_RXSTAT  ?
48081B1h  -    -    Always 0  -
48081B2h  R/W  1    W_RXSTAT  ?    "RX_RateErrorCount"
48081B3h  R/W  2    W_RXSTAT  Length>2348 error
48081B4h  R/W  3    W_RXSTAT  RXBUF Full error
48081B5h  R    4?   W_RXSTAT  ?    (R) (but seems to exist; used by firmware)
48081B6h  R/W  5    W_RXSTAT  Length=0 or Wrong FCS Error
48081B7h  R/W  6    W_RXSTAT  Packet Received Okay
                             (also increments on W_MACADDR mis-match)
                             (also increments on internal ACK packets)
                             (also increments on invalid IEEE type=3)
                             (also increments TOGETHER with 1BCh and 1BEh)
                             (not incremented on RXBUF_FULL error)
48081B8h  R/W  7    W_RXSTAT  ?
48081B9h  -    -    Always 0  -
48081BAh  R/W  8    W_RXSTAT  ?
48081BBh  -    -    Always 0  -
```

```
48081BCh  R/W  9    W_RXSTAT  WEP Error (when FC.Bit14 is set)
48081BDh  R/W  10   W_RXSTAT  ?
48081BEh  R/W  11   W_RXSTAT  (duplicated sequence control)
48081BFh  R/W  12   W_RXSTAT  ?
```

48081C4h - W_RX_COUNT (W_INTERNAL) (R)

```
0-?   Receive Okay Count (increments together with ports 48081B4h, 48081B7h)
8-?   Receive Error Count (increments together with ports 48081B3h, 48081B6h)
```

Increments when receiving a packet. Automatically reset to zero after reading.

48081D0h..1DFh - W_CMD_STAT - Multiplay Response Error Counters (R/W)

The multiplay error counters are only used when sending a multiplay command (via W_TXBUF_CMD) to any connected slaves (which must be indicated by flags located in the second halfword of the multiplay command's frame body).

```
48081D0h        Not used (always zero)
48081D1h..1DFh  Client 1..15 Response Error (increments on missing replies)
```

If one or more of those slaves fail to respond, then the corresponding error counters get incremented (at the master side). Automatically reset to zero after reading.
Unknown if these counters do also increment at the slave side?

DS Wifi Transmit Control

48080ACh - W_TXREQ_RESET - Reset Transfer Request Bits (W)

```
0-3   Reset corresponding bits in W_TXREQ_READ (0=No change, 1=Reset)
4-15  Unknown (if any)
```

Firmware writes values 01h,02h,08h,0Dh, and FFFFh.

48080AEh - W_TXREQ_SET - Set Transfer Request Bits (W)

```
0-3   Set corresponding bits in W_TXREQ_READ   (0=No change, 1=Set)
4-15  Unknown (if any)
```

Firmware writes values 01h,02h,05h,08h,0Dh.

48080B0h - W_TXREQ_READ - Get Transfer Request Bits (R)

```
0     Send W_TXBUF_LOC1  (1=Transfer, if enabled in W_TXBUF_LOC1.Bit15)
1     Send W_TXBUF_CMD   (1=Transfer, if enabled in W_TXBUF_CMD.Bit15)
2     Send W_TXBUF_LOC2  (1=Transfer, if enabled in W_TXBUF_LOC2.Bit15)
3     Send W_TXBUF_LOC3  (1=Transfer, if enabled in W_TXBUF_LOC3.Bit15)
4     Unknown (Beacon?)  (always 1, except when cleared via W_POWERFORCE)
5-15  Unknown/Not used
```

Bit0-3 can be set/reset via W_TXREQ_SET/W_TXREQ_RESET. The setting in W_TXREQ_READ remains intact even after the transfer(s) have completed.
If more than one of the LOC1,2,3 bits is set, then LOC3 is transferred first, LOC1 last.
Beacons are transferred in every Beacon Timeslot (if enabled in W_TXBUF_BEACON.Bit15).
Bit0,2,3 are automatically reset upon IRQ14 (by hardware).

48080B6h - W_TXBUSY (R)

```
0     W_TXBUF_LOC1  (1=Requested Transfer busy, or not yet started at all)
1     W_TXBUF_CMD   (1=Requested Transfer busy, or not yet started at all)
2     W_TXBUF_LOC2  (1=Requested Transfer busy, or not yet started at all)
3     W_TXBUF_LOC3  (1=Requested Transfer busy, or not yet started at all)
4     W_TXBUF_BEACON  (1=Beacon Transfer busy)
5-15  Unknown (if any)
```

Busy bits. If all three W_TXBUF_LOC's are sent, then it goes through values 0Dh,05h,01h,00h; ie. LOC3 is transferred first, LOC1 last. The register is updated upon IRQ01 (by hardware).
Bit4 is set only in Beacon Timeslots.

48080B8h - W_TXSTAT - RESULT - Status of transmitted frame (R)
For LOC1-3, this register is updated at the end of a transfer (upon the IRQ01 request), if retries occur then it is updated only after the final retry.
For BEACON, this register is updated only if enabled in W_TXSTATCNT.Bit15, and only after successful transfers (since beacon errors result in infinite retries).
For CMD, this register is updated only if enabled in W_TXSTATCNT.Bit13,14).
Bit0/1 act similar to W_IF Bit1/3, however, the W_IF Bits are set after each transmit (including retries).

```
0     One (or more) Packet has Completed (1=Yes)
      (No matter if successful, for that info see Bit1)
      (No matter if ALL packets are done, for that info see Bit12-13)
1     Packet Failed (1=Error)
2-7   Unknown/Not used
8-11  Usually 0, ...but firmware is checking for values 03h,08h,0Bh
      (gets set to 07h when transferred W_TXBUF_LOC1/2/3 did have Bit12=set)
      (gets set to 00h otherwise)
      (gets set to 03h after beacons; if enabled in W_TXSTATCNT.Bit15)
      (gets set to 08h or 0Bh after CMD; depending on W_TXSTATCNT.Bit13,14)
12-13 Packet which has updated W_TXSTAT (0=LOC1/BEACON/CMD, 1=LOC2, 2=LOC3)
14-15 Unknown/Not used
```

No idea how to reset bit0/1 once when they are set?

4808008h - W_TXSTATCNT (R/W)

```
0-12  Unknown (usually zero)
13    Update W_TXSTAT=0B01h and trigger IRQ01 after CMD transmits    (1=Yes)
14    Update W_TXSTAT=0800h and trigger IRQ01 after CMD transmits    (1=Yes)
15    Update W_TXSTAT and trigger IRQ01 after BEACON transmits (0=No, 1=Yes)
```

If both Bit13 and Bit14 are set, then Bit13 is having priority.
Note: LOC1..3 transmits are always updating W_TXSTAT and triggering IRQ01.

4808194h - W_TX_HDR_CNT - Disable Transmit Header Adjustments (R/W)

```
0     IEEE FC.Bit12 and Duration (0=Auto/whatever, 1=Manual/Wifi RAM)
1     IEEE Frame Check Sequence  (0=Auto/FCS/CRC32, 1=Manual/Wifi RAM)
2     IEEE Sequence Control      (0=Auto/W_TX_SEQNO, 1=Manual/Wifi RAM)
3-15  Always zero
```

Allows to disable automatic adjustments of the IEEE header and checksum.
Note: W_TX_SEQNO can be also disabled by W_TXBUF_LOCn.Bit13 and by TXHDR[04h].

4808210h - W_TX_SEQNO - Transmit Sequence Number (R)

```
0-11   Increments on IRQ07 (Transmit Start Interrupt)
12-15  Always zero
```

Also incremented shortly after IRQ12.
When enabled in W_TXBUF_LOCn.Bit13, this value replaces the upper 12bit of the IEEE Frame Header's Sequence Control value (otherwise, when disabled, the original value in Wifi RAM is used, and, in that case, W_TX_SEQNO is NOT incremented).
Aside from W_TXBUF_LOCn.Bit13, other ways to disable W_TX_SEQNO are: Transmit Hardware Header entry TXHDR[04h], and W_TX_HDR_CNT.Bit2.

DS Wifi Transmit Buffers

4808068h - W_TXBUF_WR_ADDR - Wifi CircBuf Write Address (R/W)

```
0     Always zero
1-12  Halfword Address in RAM for Writes via W_TXBUF_WR_DATA
13-15 Always zero
```

4808070h - W_TXBUF_WR_DATA - Wifi CircBuf Write Data (W)

```
0-15  Data to be written to address specified in W_TXBUF_WR_ADDR
```

After writing to this register, W_TXBUF_WR_ADDR is automatically incremented by 2, and, if it gets equal to W_TXBUF_GAP, then it gets additonally incremented by W_TXBUF_GAPDISP*2.

4808074h - W_TXBUF_GAP - Wifi CircBuf Write Top (R/W)

```
0     Always zero
1-12  Halfword Address
13-15 Always zero
```

4808076h - W_TXBUF_GAPDISP - CircBuf Write Offset from Top to Bottom (R/W)

```
0-11  Halfword Offset (added to; if equal to W_TXBUF_GAP)
12-15 Always zero
```

Should be "0-write_buffer_size" (wrap from end to begin), or zero (no wrapping).
Caution: On the DS-Lite, after adding it to W_TXBUF_WR_ADDR, the W_TXBUF_GAPDISP setting is destroyed (reset to 0000h) by hardware. The original DS leaves W_TXBUF_GAPDISP intact.

Note: W_TXBUF_GAP and W_TXBUF_GAPDISP may be (not TOO probably) also used by transmits via W_TXBUF_LOCn and W_TXBUF_BEACON (not tested).

4808080h - W_TXBUF_BEACON - Beacon Transmit Location (R/W)
4808090h - W_TXBUF_CMD - Multiplay Command Transmit Location (R/W)
48080A0h - W_TXBUF_LOC1 - Transmit location 1 (R/W)
48080A4h - W_TXBUF_LOC2 - Transmit location 2 (R/W)
48080A8h - W_TXBUF_LOC3 - Transmit location 3 (R/W)

```
0-11  Halfword Address of TX Frame Header in RAM
12    For LOC1-3: When set, W_TXSTAT.bit8-10 are set to 07h after transfer
                  And, when set, the transferred frame-body gets messed up?
      For BEACON: Unknown, no effect on W_TXSTAT
      For CMD: Unknown, no effect on W_TXSTAT
13    IEEE Sequence Control (0=From W_TX_SEQNO, 1=Value in Wifi RAM)
      For BEACON: Unknown (always uses W_TX_SEQNO) (no matter of bit13)
14    Unknown
15    Transfer Request (1=Request/Pending)
```

For LOC1..3 and CMD, Bit15 is automatically cleared after (or rather: during?) transfer (no matter if the transfer was successful). For Beacons, bit15 is kept unchanged since beacons are intended to be transferred repeatedly. The purpose of W_TXBUF_CMD is unknown... maybe for automatic replies...? Pictochat seems to use it for host-to-client data frames. W_TXBUF_CMD.Bit15 can be set ONLY while W_CMD_COUNT is non-zero.

48080B4h - W_TXBUF_RESET (W)

```
0     Disable LOC1    (0=No change, 1=Reset W_TXBUF_LOC1.Bit15)
1     Disable CMD     (0=No change, 1=Reset W_TXBUF_CMD.Bit15)
2     Disable LOC2    (0=No change, 1=Reset W_TXBUF_LOC2.Bit15)
3     Disable LOC3    (0=No change, 1=Reset W_TXBUF_LOC3.Bit15)
4-5   Unknown/Not used
6     Disable REPLY2  (0=No change, 1=Reset W_TXBUF_REPLY2.Bit15)
7     Disable REPLY1  (0=No change, 1=Reset W_TXBUF_REPLY1.Bit15)
8-15  Unknown/Not used
```

Firmware writes values FFFFh, 40h, 02h, xxxx, 09h, 01h, 02h, C0h.

4808084h - W_TXBUF_TIM - Beacon TIM Index in Frame Body (R/W)

```
0-7   Location of TIM parameters within Beacon Frame Body
8-15  Not used/zero
```

Usually set to 15h, that assuming that preceding Frame Body content is: Timestamp(8), BeaconInterval(2), Capability(2), SuppRatesTagLenParams(4), ChannelTagLenParam(3), TimTagLen(2); so the value points to TimParams (ie. after TimTagLen).

480806Ch - W_TXBUF_COUNT (R/W)

```
0-11  Decremented on writes to W_TXBUF_WR_DATA
12-15 Always zero
```

Triggers IRQ08 when it reaches zero, and does then stay at zero (without further decrementing, and without generating further IRQs).
Note: Not affected by (accidental) reads from write-only W_TXBUF_WR_DATA.

DS Wifi Transmit Errors

Automatic ACKs
Transmit errors occur on missing ACKs. The NDS hardware is automatically responding with an ACK when receiving a packet (if it has been addressed to the receipients W_MACADDR setting). And, when sending a packet, the NDS hardware is automatically checking for ACK responses.
The only exception are packets that are sent to group addresses (ie. Bit0 of the 48bit MAC address being set to "1", eg. Beacons sent to FF:FF:FF:FF:FF:FF), the receipient(s) don't need to respond to such packets, and the sender always passes okay without checking for ACKs.

480802Ch - W_TX_RETRYLIMIT (R/W)
Specifies the maximum number of retries on Transmit Errors (eg. 07h means one initial transmit, plus up to 7 retries, ie. max 8 transmits in total).

```
0-7   Retry Count (usually 07h)
8-15  Unknown     (usually 07h)
```

The Retry Count value is decremented on each Error (unless it is already 00h). There's no automatic reload, so W_TX_RETRYLIMIT should be reinitialized by software prior to each transmit (or, actually, there IS a reload?).
When sending multiple packets (by setting more than one bit with W_TXREQ_SET), then the first packet may eat-up all retries, leaving only a single try to the other packet(s).

48081C0h - W_TX_ERR_COUNT - TransmitErrorCount (R/W)

```
0-7   TransmitErrorCount
8-15  Always zero
```

Increments on Transmit Errors. Automatically reset to zero after reading.
IRQ03 triggered when W_TX_ERR_COUNT is incremented (for NON-beacons ONLY).
IRQ05 triggered when W_TX_ERR_COUNT > 7Fh (happens INCLUDING for beacons).

Error Notification
Transmit Errors can be sensed via W_TX_ERR_COUNT, IRQ03, IRQ05, TX Hardware Header entry [00h], and W_TXSTAT.Bit1.

W_TXBUF_BEACON Errors
As the name says, W_TXBUF_BEACON is intended for sending Beacons to group addresses (which do not require to respond by ACKs). So, transmit errors would occur only when mis-using W_TXBUF_BEACON to send packets to individual addresses, but the W_TXBUF_BEACON error handling isn't fully implemented: First of, W_TX_RETRYLIMIT isn't used, instead, W_TXBUF_BEACON errors will result in infinite retries. Moreover, W_TXBUF_BEACON errors seem to increment W_TX_ERR_COUNT, but without generating IRQ03, however, IRQ05 is generated when W_TX_ERR_COUNT>7Fh.

Other Errors
The NDS transmit hardware seems to do little error checking on the packet headers. The only known error-checked part is byte [04h] in the TX hardware header (which must be 00h, 01h, or 02h). Aside from that, when sent to a group address, it is passing okay even with invalid IEEE type/subtypes, and even with Length/Rate entries set to zero. However, when sending such data to an individual address, the receiving NDS won't respond by ACKs.

Note
Received ACKs aren't stored in WifiRAM (or, possibly, they ARE stored, but without advancing

W_RXBUF_WRCSR, so that the software won't see them, and so that they will be overwritten by the next packet).

DS Wifi Status

480819Ch - W_RF_PINS - Status of RF-Chip Control Signals (R)

```
  0    Reportedly "carrier sense" (maybe 1 during RX.DTA?) (usually 0)
  1    TX.MAIN (RFU.Pin17) Transmit Data Phase          (0=No, 1=Active)
  2    Unknown (RFU.Pin3)  Seems to be always high      (Always 1=high?)
  3-5  Not used                                         (Always zero)
  6    TX.ON   (RFU.Pin14) Transmit Preamble+Data Phase (0=No, 1=Active)
         Uhhh, no that seems to be still wrong...
         Bit6 is often set, even when not transmitting anything...
  7    RX.ON   (RFU.Pin15) Receive Mode                 (0=No, 1=Enable)
  8-15 Not used                                         (Always zero)
```

Physical state of the RFU board's RX/TX pins. Similar to W_RF_STATUS.

4808214h - W_RF_STATUS - Current Transmit/Receive State (R)

```
  0-3  Current Transmit/Receive State:
        0 = Initial Value on power-up (before raising W_MODE_RST.Bit0)
        1 = RX Mode enabled (waiting for incoming data)
        2 = Switching from RX to TX (takes a few clock cycles)
        3 = TX Mode active  (sending preamble and data)
        4 = Switching from TX to RX (takes a few clock cycles)
        5 = Unknown, firmware checks for that value (maybe RX busy)
        6 = Unknown, firmware checks for that value (maybe RX busy)
        9 = Idle (upon IRQ13, and upon raising W_MODE_RST.Bit0)
        ----
        5 = Receive ACK phase ?
        6 =
        7 =
        8 = Multiplay related ? (when sending through W_TXBUF_CMD ?)
  4-15 Always zero?
```

Numeric Status Code. Similar to W_RF_PINS.

4808268h - W_RXTX_ADDR - Current Receive/Transmit Address (R)

```
  0-11   Halfword address
  12-15  Always zero
```

Indicates the halfword that is currently transmitted or received. Can be used by Start Receive IRQ06 handler to determine how many halfwords of the packet have been already received (allowing to pre-examine portions of the packet header even when the whole packet isn't fully received). Can be also used in Transmit Start IRQ07 handler to determine which packet is currently transmitted.

DS Wifi Timers

48080E8h - W_US_COUNTCNT - Microsecond counter enable (R/W)

```
  0     Counter Enable (0=Disable, 1=Enable)
  1-15  Always zero
```

Activates W_US_COUNT, and also W_BEACONCOUNT1 and W_BEACONCOUNT2 (which are decremented when lower 10bit of W_US_COUNT wrap from 3FFh to 000h). Note: W_POWER_US must be enabled, too.

48080F8h - W_US_COUNT0 - Microsecond counter, bits 0-15 (R/W)
48080FAh - W_US_COUNT1 - Microsecond counter, bits 16-31 (R/W)
48080FCh - W_US_COUNT2 - Microsecond counter, bits 32-47 (R/W)
48080FEh - W_US_COUNT3 - Microsecond counter, bits 48-63 (R/W)

```
  0-63  Counter Value in microseconds (incrementing)
```
Clocked by the 22.00MHz oscillator on the RFU board (ie. not by the 33.51MHz system clock). The 22.00MHz are divided by a 22-step prescaler.

48080EAh - W_US_COMPARECNT - Microsecond compare enable (R/W)
```
  0     Compare Enable (0=Disable, 1=Enable) (IRQ14/IRQ15)
  1     Force IRQ14    (0=No, 1=Force Now)   (Write-only)
  2-15  Always zero
```
Activates IRQ14 on W_US_COMPARE matches, and IRQ14/IRQ15 on W_BEACONCOUNT1.

48080F0h - W_US_COMPARE0 - Microsecond compare, bits 0-15 (R/W)
48080F2h - W_US_COMPARE1 - Microsecond compare, bits 16-31 (R/W)
48080F4h - W_US_COMPARE2 - Microsecond compare, bits 32-47 (R/W)
48080F6h - W_US_COMPARE3 - Microsecond compare, bits 48-63 (R/W)
```
  0     Always zero... firmware writes 1 though (maybe write-only flag?)
  1-9   Always zero
  10-63 Compare Value in milliseconds (aka microseconds/1024)
```
Triggers IRQ14 (see IRQ14 notes below) when W_US_COMPARE matches W_US_COUNT.
Usually set to FFFFFFFFFFFFC00h (ie. almost/practically never). Instead, IRQ14 is usually derived via W_BEACONCOUNT1.

480811Ch - W_BEACONCOUNT1 (R/W)
Triggers IRQ14 and IRQ15 (see IRQ14/IRQ15 notes below) when it reaches 0000h (if W_PRE_BEACON is non-zero, then IRQ15 occurs that many microseconds in advance).
```
  0-15  Decrementing Millisecond Counter (reloaded with W_BEACONINT upon IRQ14)
```
Set to W_BEACONINT upon IRQ14 events (unlike the other W_US_COMPARE related actions, this is done always, even if W_US_COMPARECNT is zero).
When reaching 0000h, it is immediately reloaded (as for US_COUNT matches), so the counting sequence is ..,3,2,1,BEACONINT,.. (not 3,2,1,ZERO,BEACONINT).

4808134h - W_BEACONCOUNT2 - Post-Beacon Counter (R/W)
```
  0-15  Decrementing Millisecond Counter (reloaded with FFFFh upon IRQ14)
```
Triggers IRQ13 when it reaches 0000h (no matter of W_US_COMPARECNT), and does then stay fixed at 0000h (without any further decrement/wrapping to FFFFh).
Set to FFFFh upon IRQ14 (by hardware), the IRQ14 handler should then adjust the register (by software) by adding the Tag DDh Beacon header's Stepping value (usually 000Ah) to it.
Seems to be used to indicate beacon transmission time (possible including additional time being reserved for responses)?

480808Ch - W_BEACONINT - Beacon Interval (R/W)
Reload value for W_BEACONCOUNT1.
```
  0-9   Frequency in milliseconds of beacon transmission
  10-15 Always zero
```
Should be initialized randomly to 0CEh..0DEh or so. The random setting reduces risk of repeated overlaps with beacons from other hosts.

4808110h - W_PRE_BEACON - Pre-Beacon Time (R/W)
```
  0-15  Pre-Beacon Time in microseconds (static value, ie. NOT decrementing)
```
Allows to define the distance between IRQ15 and IRQ14. The setting doesn't affect the IRQ14 timing (which occurs at the W_BEACONCOUNT1'th millisecond boundary), but IRQ15 occurs in advance (at the W_BEACONCOUNT1'th millisecond boundary minus W_PRE_BEACON microseconds). If W_PRE_BEACON is zero, then both IRQ14 and IRQ15 occur exactly at the same time.

4808088h - W_LISTENCOUNT - Listen Count (R/W)
```
  0-7   Decremented by hardware at IRQ14 events (ie. once every beacon)
  8-15  Always zero
```

Reload occurs immediately BEFORE decrement, ie. with W_LISTENINT=04h, it will go through values 03h,02h,01h,00h,03h,02h,01h,00h,etc.

480808Eh - W_LISTENINT - Listen Interval (R/W)

```
  0-7   Listen Interval, counted in beacons (usually 02h)
  8-15  Always zero
```

Reload value for W_LISTENCOUNT.

480810Ch - W_CONTENTFREE (R/W)

```
  0-15  Decrementing microsecond counter
```

Operated always (no matter of W_US_COUNTCNT).
Once when it has reached 0000h, it seems to stay fixed at 0000h.
"[Set to the remaining duration of contention-free period when
receiving beacons - only *really* necessary for powersaving mode]"

IRQ13 Notes (Post-Beacon Interrupt)

IRQ13 is generated by W_BEACONCOUNT2. It's simply doing:

```
  W_IF.Bit13=1      ;interrupt request
```

If W_POWER_TX.Bit1=0, then additionally enter sleep mode:

```
  [4808034h]=0002h ;W_INTERNAL   ;(similar to W_POWERFORCE=8001h)
  [480803Ch]=02xxh ;W_POWERSTATE ;(W_TXREQ_READ.Bit4 is kept intact though)
  [480819Ch]=0046h ;W_RF_PINS.7=0;disable receive (enter idle mode) (RX.ON=Low)
  [4808214h]=0009h ;W_RF_STATUS=9;indicate idle mode
```

Unlike for IRQ14/IRQ15, that's done no matter of W_US_COMPARECNT.

IRQ14 Notes (Beacon Interrupt)

IRQ14 is generated by W_US_COMPARE, and by W_BEACONCOUNT1.
Aside from just setting the IRQ flag in W_IF, the hardware does:

```
  W_BEACONCOUNT1=W_BEACONINT                          ;next IRQ15/IRQ14
  (Above is NOT done when IRQ14 was forced via W_US_COMPARECNT.Bit1)
```

If W_US_COMPARECNT is 1, then the hardware does additionally:

```
  (Below IS ALSO DONE when IRQ14 was forced via W_US_COMPARECNT.Bit1)
  W_IF.Bit14=1
  W_BEACONCOUNT2=FFFFh ;about 64 secs (ie. almost never) ;next IRQ13 ("never")
  W_TXREQ_READ=W_TXREQ_READ AND FFF2h
  if W_TXBUF_BEACON.15 then W_TXBUSY.Bit4=1
  if W_LISTENCOUNT=00h then W_LISTENCOUNT=W_LISTENINT
  W_LISTENCOUNT=W_LISTENCOUNT-1
```

If W_TXBUF_BEACON.Bit15=1, then following is done shortly after IRQ14:

```
  W_RF_PINS.Bit7=0  ;disable receive (RX.ON=Low)
  W_RF_STATUS=2     ;indicate switching from RX to TX mode
```

If W_TXBUF_BEACON.Bit15=1, then following is done a bit later:

```
  W_RF_PINS.Bit6=1  ;transmit preamble start (TX.ON=High)
  W_RF_STATUS=3     ;indicate TX mode
```

The IRQ14 handler should then do the following (by software):

```
  W_BEACONCOUNT2 = W_BEACONCOUNT2 + TagDDhSteppingValue  ;next IRQ13
```

For using only ONE of the two IRQ14 sources: W_BEACONCOUNT1 can be disabled by setting both W_BEACONCOUNT1 and W_BEACONINT to zero. W_US_COMPARE can be sorts of "disabled" by setting it to value distant from W_US_COUNT, such like compare=count-400h.

IRQ07 Notes (Transmit Start Data; occurs after preamble)

```
  W_IF.Bit7=1       ;interrupt request
  W_RF_PINS.Bit1=1  ;start data transfer (preamble finished now) (TX.MAIN=High)
```

Below only if packet was sent through W_TXBUF_BEACON, or if it was sent via W_TXBUF_LOCn, with W_TXBUF_LOCn.Bit13 being zero:

```
  [TXBUF...] = W_TX_SEQNO*10h   ;auto-adjust IEEE Sequence Control
  W_TX_SEQNO=W_TX_SEQNO+1       ;increase sequence number
```

IRQ01 Notes (Transmit Done)
The following happens shortly before IRQ01:
```
  W_RF_PINS.Bit6=0  ;disable TX (TX.ON=Low)
  W_RF_STATUS=4     ;indicate switching from TX to RX mode
```
Then, upon IRQ01, the following happens:
```
  W_IF.Bit1=1       ;interrupt request
  W_RF_PINS.Bit1=0  ;disable TX (TX.MAIN=Low)
  W_RF_PINS.Bit7=1  ;enable RX (RX.ON=High)
  W_RF_STATUS=1     ;indicate RX mode
```

IRQ15 Notes (Pre-Beacon Interrupt)
IRQ15 is generated via W_BEACONCOUNT1 and W_PRE_BEACON. It's simply doing:
```
  if W_US_COMPARECNT=1 then W_IF.Bit15=1
```
If W_POWER_TX.Bit0=1, then additionally wakeup from sleep mode:
```
  W_RF_PINS.Bit7=1  ;enable RX (RX.ON=High) ;\gets set like so a good while
  W_RF_STATUS=1     ;indicate RX mode       ;/after IRQ15 (but not immediately)
```

Beacon IRQ Sequence
```
  IRQ15  Pre-Beacon  (beacon will be transferred soon)
  IRQ14  Beacon      (beacon will be transferred very soon) (carrier starts)
  IRQ07  Tx Start    (beacon transfer starts) (if enabled in W_TXBUF_BEACON.15)
  IRQ01  Tx End      (beacon transfer done) (if enabled in W_TXSTATCNT.15)
  IRQ13  Post-Beacon (beacon transferred) (unless next IRQ14 occurs earlier)
```
That, for tranmitting beacons. (For receiving, IRQ07/IRQ01 would be replaced by Rx IRQ's, provided that a remote unit is sending beacons).

DS Wifi Multiplay Master

These registers are used for multiplay host-to-client (aka master to slave) commands.

48080EEh - W_CMD_COUNTCNT (R/W)
```
  0     Enable W_CMD_COUNT (0=Disable, 1=Enable)
  1-15  Always Zero
```

4808118h - W_CMD_COUNT (R/W)
```
  0-15  Decremented once every 10 microseconds (Stopped at 0000h)
```
Written by firmware. Firmware IRQ14 handler checks for read value<=0Ah.
When it reaches zero, W_TXBUF_CMD is transferred (if enabled in W_TXBUF_CMD.Bit15, and in W_TXREQ_READ.Bit1), it does then trigger two (!) transfer start interrupts (IRQ07), transfer end is then indicated by a single IRQ12, optionally (when enabled in W_TXSTATCNT, IRQ01 (transfer done) is additionally generated (simultaneously with above IRQ12).
NOPE, above isn't quite right..... when W_CMD_COUNT is set to a very small value, then ONLY IRQ12 is triggered (so it might specify the duration during which the IRQ07's for W_TXBUF_CMD are allowed?)

48080C0h - W_CMD_TOTALTIME - (R/W)
```
  0-15  Duration per ALL slave response packet(s) in microseconds
```
Before sending a MASTER packet, this port should be set to the same value as the MASTER packet's IEEE header's Duration/ID entry.

48080C4h - W_CMD_REPLYTIME - (R/W)
```
  0-15  Duration per SINGLE slave response packet in microseconds
```
Before sending a MASTER packet, this port should be set to the expected per slave response time.
Note: Nintendo's multiboot/pictochat code is also putting this value in the 1st halfword of the MASTER packet's frame body.

At 2MBit/s transfer rate, the values should be set up sorts of like so:

```
  master_time = (master_bytes*4)+(60h)     ;60h = 96 decimal = short preamble
  slave_time = (slave_bytes*4)+(0D0h..0D2h)
  all_slave_time = (EAh..F0h)+(slave_time+0Ah)*num_slaves
  txhdr[2]   = slave_bits      ;hardware header (*)
  ieee[2]    = all_slave_time  ;ieee header (duration/id)
  body[0]    = slave_time      ;duration per slave (for multiboot/pictochat)
  body[2]    = slave_bits      ;frame body -- required (*)
  [48080C0h] = all_slave_time  ;
  [48080C4h] = slave_time      ;duration per slave
  [4808118h] = (388h+(num_slaves*slave_time)+master_time+32h)/10
  [4808090h] = 8000h+master_packet_address   ;start transmit
```

With the byte values counting the ieee frame header+body+fcs.
(*) The hardware doesn't actually seem to use the "slave_bits" entry in the hardware header, instead, it is using the "slave_bits" entry in the frame body(!)

DS Wifi Multiplay Slave

These registers are used for multiplay client-to-host (aka slave to master) responses.

4808094h - W_TXBUF_REPLY1 - Multiplay Response Transmit Location 1 (R/W)

```
  0-11  Halfword address
  12-14 Unknown (the bits can be set, ie. they DO exist)
  15    Enable
```

Response packet address. The register setting probably doesn't directly affect the hardware, it's sole purpose seems to initialize 4808098h (see there).

4808098h - W_TXBUF_REPLY2 - Multiplay Response Transmit Location 2 (R)

```
  0-11  Halfword address
  12-14 Unknown (the bits can be set, ie. they DO exist)
  15    Enable
```

This register seems to contain the actual response packet address. However, since it's read-only, software cannot set it directly. Instead, software must write the address to 4808094h, and then latch it from 4808094h to 4808098h (via. W_RXCNT.Bit7).

Notes
Not sure if there's also auto-latching (similar to manual W_RXCNT.Bit7)?
Unknown if W_TXBUF_REPLY2.Bit15 is automatically reset after transfer?
Not sure if/how the hardware determines WHEN to send reply packets (eg. it should NOT send them after receiving Beacons) (eventually the Start Receive IRQ handler must examine the incoming packet, and then software must decide if it wants to respond by sending the reply) (if there are multiple slaves, the response order is probably automatically handled in respect to the local W_AID_LOW setting) (although, if, for example, ONLY slave 5 exists, then it ought to know that slave 5 is the <first> slave; that might happen if slave 1..4 have left the communication; that, unless the slaves would be automatically renumbered by software (?), so slave 5 would be become slave 1). Some of the Unknown Registers (namely Ports W_X_244h and W_X_228h) are probably also related to the REPLY function.

DS Wifi Configuration Ports

4808120h - W_CONFIG_120h (R/W) ;init from firmware[04Ch] ;81ff 0048->SAME
4808122h - W_CONFIG_122h (R/W) ;init from firmware[04Eh] ;ffff 4840->SAME
4808124h - W_CONFIG_124h (R/W) ;init from firmware[05Eh] ;ffff 0000->0032
4808128h - W_CONFIG_128h (R/W) ;init from firmware[060h] ;ffff 0000->01F4

4808130h - W_CONFIG_130h (R/W) ;init from firmware[054h] ;0fff 0142->0140
4808132h - W_CONFIG_132h (R/W) ;init from firmware[056h] ;8fff 8064->SAME
4808140h - W_CONFIG_140h (R/W) ;init from firmware[058h] ;ffff 0000->E0E0
4808142h - W_CONFIG_142h (R/W) ;init from firmware[05Ah] ;ffff 2443->SAME
4808144h - W_CONFIG_144h (R/W) ;init from firmware[052h] ;00ff 0042->SAME
4808146h - W_CONFIG_146h (R/W) ;init from firmware[044h] ;00ff 0016->0002
4808148h - W_CONFIG_148h (R/W) ;init from firmware[046h] ;00ff 0016->0017
480814Ah - W_CONFIG_14Ah (R/W) ;init from firmware[048h] ;00ff 0016->0026
480814Ch - W_CONFIG_14Ch (R/W) ;init from firmware[04Ah] ;ffff 162C->1818
4808150h - W_CONFIG_150h (R/W) ;init from firmware[062h] ;ff3f 0204->0101
4808154h - W_CONFIG_154h (R/W) ;init from firmware[050h] ;7a7f 0058->SAME
These ports are to be initialized from firmware settings.
Above comments show the R/W bits (eg. 81FFh means bit15 and bit8-0 are R/W, bit14-9 are always zero), followed by the initial value on Reset (eg. 0048h), followed by new value after initialization from firmware settings (eg. 0032h, or SAME if the Firmware value is equal to the Reset value; these values seem to be identical in all currently existing consoles).
Note: Firmware part4 changes W_CONFIG_124h to C8h, and W_CONFIG_128h to 7D0h, and W_CONFIG_150h to 202h, and W_CONFIG_140h depending on tx rate and preamble:

```
W_CONFIG_140h = firmware[058h]+0202h             ;1Mbit/s
W_CONFIG_140h = firmware[058h]+0202h-6161h       ;2Mbit/s with long preamble
W_CONFIG_140h = firmware[058h]+0202h-6161h-6060h ;2Mbit/s with short preamble
```

48080ECh - W_CONFIG_0ECh (R/W) ;firmware writes 3F03h (same as on power-up)
48080D4h - W_CONFIG_0D4h (R/W) ;firmware writes 0003h (affectd by W_MODE_RST)
48080D8h - W_CONFIG_0D8h (R/W) ;firmware writes 0004h (same as on power-up)
4808254h - W_CONFIG_254h (?) ;firmware writes 0000h (read: EEEEh on DS-Lite)
Firmware just initializes these ports with fixed values, without further using them after initialization.

48080DAh - W_RX_LEN_CROP (R/W) ;firmware writes 0602h (same as on power-up)

```
0-7   Decrease RX Length by N halfwords for Non-WEP packets (usually 2)
8-15  Decrease RX Length by N halfwords for WEP packets     (usually 6)
```

Used to exclude the FCS (and WEP IV+ICV) from the length entry in the Hardware RX Header.

DS Wifi Baseband Chip (BB)

BB-Chip Mitsumi MM3155 (DS), or BB/RF-Chip Mitsumi MM3218 (DS-Lite)

4808158h - W_BB_CNT - Baseband serial transfer control (W)

```
0-7   Index     (00h-68h)
8-11  Not used  (should be zero)
12-15 Direction (5=Write BB_WRITE to Chip, 6=Read from Chip to BB_READ)
```

Transfer is started after writing to this register.

480815Ah - W_BB_WRITE - Baseband serial write data (W)

```
0-7   Data to be sent to chip (by following W_BB_CNT transfer)
8-15  Not used (should be zero)
```

480815Ch - W_BB_READ - Baseband serial read data (R)

```
0-7   Data received from chip (from previous W_BB_CNT transfer)
8-15  Not used (always zero)
```

480815Eh - W_BB_BUSY - Baseband serial busy flag (R)

```
0     Transfer Busy (0=Ready, 1=Busy)
1-15  Always zero
```

Used to sense transfer completion after writes to W_BB_CNT.
Not sure if I am doing something wrong... but the busy flag doesn't seem to get set immediately after W_BB_CNT writes, and works only after waiting a good number of clock cycles?

4808160h - W_BB_MODE (R/W)

```
0-7   Always zero
8     Unknown (usually 1) (no effect no matter what setting?)
9-13  Always zero
14    Unknown (usually 0) (W_BB_READ gets unstable when set)
15    Always zero
```

This register is initialized by firmware bootcode - don't change.

4808168h - W_BB_POWER (R/W)

```
0-3   Disable whatever   (usually 0Dh=disable)
4-14  Always zero
15    Disable W_BB_ports (usually 1=Disable)
```

Must be set to 0000h before accessing BB registers.

Read-Write-Ability of the BB-Chip Mitsumi MM3155 registers (DS)

```
Index    Num Dir Expl.
00h        1 R   always 6Dh (R) (Chip ID)
01h..0Ch  12 R/W 8bit R/W
0Dh..12h   6 -   always 00h
13h..15h   3 R/W 8bit R/W
16h..1Ah   5 -   always 00h
1Bh..26h  12 R/W 8bit R/W
27h        1 -   always 00h
28h..4Ch     R/W 8bit R/W
4Dh        1 R   always 00h or BFh (depending on other regs)
4Eh..5Ch     R/W 8bit R/W
5Dh        1 R   always 01h (R)
5Eh..61h     -   always 00h
62h..63h   2 R/W 8bit R/W
64h        1 R   always FFh or 3Fh (depending on other regs)
65h        1 R/W 8bit R/W
66h        1 -   always 00h
67h..68h   2 R/W 8bit R/W
69h..FFh     -   always 00h
```

Read-Write-Ability of the BB/RF-Chip Mitsumi MM3218 (DS-Lite)

Same as above. Except that reading always seems to return [5Dh]=00h. And, for whatever reason, Nintendo initializes DS-Lite registers by writing [00h]=03h and [66h]=12h. Nethertheless, the registers always read as [00h]=6Dh and [66h]=00h, ie. same as on original DS.

Important BB Registers

Registers 0..68h are initialized by firmware bootcode, and (most) of these settings do not need to be changed by other programs, except for:

```
Addr Initial Meaning
01h 0x9E    [unsetting/resetting bit 7 initializes/resets the system?]
02h         unknown (firmware is messing with this register)
06h         unknown (firmware is messing with this register, too)
13h 0x00    CCA operation - criteria for receiving
                  0=only use Carrier Sense (CS)
                  1=only use Energy Detection (ED)
                  2=receive if CS OR ED
                  3=receive only if CS AND ED
1Eh 0xBB    see change channels flowchart (Ext. Gain when RF[09h].bit16=0)
35h 0x1F    Energy Detection (ED) criteria
            value 0..61 (representing energy levels of -60dBm to -80dBm)
```

DS Wifi RF Chip

RF-Chip RF9008 (compatible to RF2958 from RF Micro Devices, Inc.) (Original DS)
BB/RF-Chip Mitsumi MM3218 (DS-Lite)

480817Ch - W_RF_DATA2 - RF chip serial data/transfer enable (R/W)
For Type2 (ie. firmware[040h]<>3):
```
  0-1   Upper 2bit of 18bit data
  2-6   Index   (00h..1Fh) (firmware uses only 00h..0Bh)
  7     Command (0=Write data, 1=Read data)
  8-15  Should be zero (not used with 24bit transfer)
```
For Type3 (ie. firmware[040h]=3):
```
  0-3   Command (5=Write data, 6=Read data)
  4-15  Should be zero (not used with 20bit transfer)
```
Writing to this register starts the transfer.

480817Eh - W_RF_DATA1 - RF chip serial data (R/W)
For Type2 (ie. firmware[040h]<>3):
```
  0-15  Lower 16bit of 18bit data
```
For Type3 (ie. firmware[040h]=3):
```
  0-7   Data (to be written to chip) (or being received from chip)
  8-15  Index (usually 00h..28h) (index 40h..FFh are mirrors of 00h..3Fh)
```
This value should be set before setting W_RF_DATA2.

4808180h - W_RF_BUSY - RF chip serial busy flag (R)
```
  0     Transfer Busy (0=Ready, 1=Busy)
  1-15  Always zero
```
Used to sense transfer completion after writes to W_RF_DATA2.

4808184h - W_RF_CNT - RF chip serial control (R/W)
```
  0-5   Transfer length (init from firmware[041h].Bit0-5)
  6-7   Always zero
  8     Unknown         (init from firmware[041h].Bit7)
  9-13  Always zero
  14    Unknown         (usually 0)
  15    Always zero
```
This register is initialized by firmware bootcode - don't change.
Usually, Type2 has length=24bit and flag=0. Type3 uses length=20bit and flag=1.

Caution For Type2 (ie. firmware[040h]<>3)
Before accessing Type2 RF Registers, first BB[01h] must have been properly initialized (ie. BB[01h].Bit7 must have been toggled from 0-to-1).

DS Wifi RF9008 Registers

RF9008 (RF2958 compatible)
2.4GHz Spread-Spectrum Transceiver - RF Micro Devices, Inc.

RF chip data (Type2) (initial NDS settings from firmware, example)
```
  Firmware   Index   Data
  (24bit)    (4bit)  (18bit)
  00C007h  =  00h  + 0C007h ;-also set to 0C008h for power-down
  129C03h  =  04h  + 29C03h
  141728h  =  05h  + 01728h ;\these are also written when changing channels
```

```
  1AE8BAh  =  06h  + 2E8BAh ;/
  1D456Fh  =  07h  + 1456Fh
  23FFFAh  =  08h  + 3FFFAh
  241D30h  =  09h  + 01D30h ;-bit10..14 should be also changed per channel?
  """"50h  =  """  + """50h ;firmware v5 and up uses narrower tx filter
  280001h  =  0Ah  + 00001h
  2C0000h  =  0Bh  + 00000h
  069C03h  =  01h  + 29C03h
  080022h  =  02h  + 00022h
  0DFF6Fh  =  03h  + 1FF6Fh
```

RF[00h] - Configuration Register 1 (CFG1) (Power on: 00007h)

```
  17-16 Reserved, program to zero (0)
  15-14 Reference Divider Value (0=Div2, 1=Div3, 2=Div44, 3=Div1)
  3     Sleep Mode Current      (0=Normal, 1=Very Low)
  2     RF VCO Regulator Enable (0=Disable, 1=Enable)
  1     IF VCO Regulator Enable (0=Disable, 1=Enable)
  0     IF VGA Regulator Enable (0=Disable, 1=Enable)
```

RF[01h] - IF PLL Register 1 (IFPLL1) (Power on: 09003h)

```
  17    IF PLL Enable                      (0=Disable, 1=Enable)
  16    IF PLL KV Calibration Enable       (0=Disable, 1=Enable)
  15    IF PLL Coarse Tuning Enable        (0=Disable, 1=Enable)
  14    IF PLL Loop Filter Select          (0=Internal, 1=External)
  13    IF PLL Charge Pump Leakage Current (0=Minimum value, 1=2*Minimum value)
  12    IF PLL Phase Detector Polarity     (0=Positive, 1=Negative)
  11    IF PLL Auto Calibration Enable     (0=Disable, 1=Enable)
  10    IF PLL Lock Detect Enable          (0=Disable, 1=Enable)
  9     IF PLL Prescaler Modulus           (0=4/5 Mode, 1=8/9 Mode)
  8-4   Reserved, program to zero (0)
  3-0   IF VCO Coarse Tuning Voltage       (N=Voltage*16/VDD)
```

RF[02h] - IF PLL Register 2 (IFPLL2) (Power on: 00022h)

```
  17-16 Reserved, program to zero (0)
  15-0  IF PLL divide-by-N value
```

RF[03h] - IF PLL Register 3 (IFPLL3) (Power on: 1FF78h)

```
  17    Reserved, program to zero (0)
  16-8  IF VCO KV Calibration, delta N value (signed)  ;DeltaF=(DN/Fr)
  7-4   IF VCO Coarse Tuning Default Value
  3-0   IF VCO KV Calibration Default Value
```

RF[04h] - RF PLL Register 1 (RFPLL1) (Power on: 09003h)

```
  17-10 Same as for RF[01h] (but for RF, not for IF)
  9     RF PLL Prescaler Modulus (0=8/9 Mode, 1=8/10 Mode)
  8-0   Same as for RF[01h] (but for RF, not for IF)
```

RF[05h] - RF PLL Register 2 (RFPLL2) (Power on: 01780h)

```
  17-6  RF PLL Divide By N Value
  5-0   RF PLL Numerator Value (Bits 23-18)
```

RF[06h] - RF PLL Register 3 (RFPLL3) (Power on: 00000h)

```
  17-0  RF PLL Numerator Value (Bits 17-0)
```

RF[07h] - RF PLL Register 4 (RFPLL4) (Power on: 14578h)

```
  17-10 Same as for RF[03h] (but for RF, not for IF) ;and, DN=(deltaF/Fr)*256
```

RF[08h] - Calibration Register 1 (CAL1) (Power on: 1E742h)

```
  17-13  VCO1 Warm-up Time  ;TVCO1=(approximate warm-up time)*(Fr/32)
  12-8   VCO1 Tuning Gain Calibration ;TLOCK1=(approximate lock time)*(Fr/128)
```

```
  7-3    VCO1 Coarse Tune Calibration Reference  ;VALUE=(average time)*(Fr/32)
  2-0    Lock Detect Resolution (0..7)
```

RF[09h] - TXRX Register 1 (TXRX1) (Power on: 00120h)

```
  17    Receiver DC Removal Loop          (0=Enable DC Removal Loop, 1=Disable)
  16    Internal Variable Gain for VGA  (0=Disable/External, 1=Enable/Internal)
  15    Internal Variable Gain Source (0=From TXVGC Bits, 1=From Power Control)
  14-10 Transmit Variable Gain Select (TXVGC)   (0..1Fh = High..low gain)
  9-7   Receive Baseband Low Pass Filter     (0=Wide Bandwidth, 7=Narrow)
  6-4   Transmit Baseband Low Pass Filter    (0=Wide Bandwidth, 7=Narrow)
  3     Mode Switch            (0=Single-ended mode, 1=Differential mode)
  2     Input Buffer Enable TX (0=Input Buffer Controlled by TXEN, 1=By BBEN)
  1     Internal Bias Enable   (0=Disable/External, 1=Enable/Internal)
  0     TX Baseband Filters Bypass        (0=Not Bypassed, 1=Bypassed)
```

RF[0Ah] - Power Control Register 1 (PCNT1) (Power on: 00000h)

```
  17-15 Select MID_BIAS Level                          (1.6V through 2.6V)
  14-9  Desired output power at antenna                (N*0.5dBm)
  8-3   Power Control loop-variation-adjustment Offset (signed, N*0.5dB)
  2-0   Desired delay for using a single TX_PE line    (N*0.5us)
```

RF[0Bh] - Power Control Register 2 (PCNT2) (Power on: 00000h)

```
  17-12 Desired MAX output power when PABIAS=MAX=2.6V (N*0.5dBm)
  11-6  Desired MAX output power when PABIAS=MID_BIAS (N*0.5dBm)
  5-0   Desired MAX output power when PABIAS=MIN=1.6V (N*0.5dBm)
```

RF[0Ch] - VCOT Register 1 (VCOT1) (Power on: 00000h)

```
  17    IF VCO Band Current Compensation (0=Disable, 1=Enable)
  16    RF VCO Band Current Compensation (0=Disable, 1=Enable)
  15-0  Reserved, program to zero (0)
```

RF[0Dh..1Ah] - N/A (Power on: 00000h)

```
  Not used.
```

RF[1Bh] - Test Register 1 (TEST) (Power on: 0000Fh)

```
  17-0  This is a test register for internal use only.
```

RF[1Ch..1Eh] - N/A (Power on: 00000h)

```
  Not used.
```

RF[1Fh] - Reset Register (Power on: 00001h)

```
  17-0  Don't care (writing any value resets the chip)
```

DS Wifi Unknown Registers

480800Ah - W_X_00Ah (R/W)

```
  0-15  Unknown (usually zero)
```

"[bit7 - ingore rx duplicates]" <--- that is NOT correct (no effect).
Firmware writes 0000h to it. That, done many times. So, eventually some bits in this register are automatically set by hardware in whatever situations, otherwise repeatedly writing 0000h to it would be kinda useless...?

Below Ports W_X_244h and W_X_228h might be related to deciding when to send multiplay replies...?

4808244h - W_X_244h (R/W) x ffff [0000] (used by firmware part4)
Unknown. Seems to be W_IF/W_IE related. Firmware sets 4808Port 244h bits 6,7,12 to 1-then-0 upon

IRQ06,IRQ07,IRQ12 respectively.

4808228h - W_X_228h (W) fixx [0000] (used by firmware part4) (bit3)
Unknown. Firmware writes 8-then-0 (done in IRQ06 handler, after Port 4808244h access).

Below Ports 48081A0h, 48081A2h, 48081A4h are somehow related to BB[02h]...

48081A0h - W_X_1A0h - (R/W) -933 [0000]
```
0-1   Unknown
2-3   Always zero
4-5   Unknown
6-7   Always zero
8     Unknown
9-10  Always zero
11    Unknown
12-15 Always zero
```
Firmware writes values 000h, 823h. Seems to be power-related. The following experimental code toggles RXTX.ON (RFU.Pin4): "x=0 / @@lop: / [48081A0h]=x / [4808036h]=0 / x=x XOR 3 / wait_by_loop(1000h) / b @@lop".
Also, writing to port 48081A0h affects ports 4808034h, 480819Ch, 480821Ch, and 48082A2h.

48081A2h - W_X_1A2h - (R/W) ---3 [0001] (used by firmware part4)
```
0-1   Unknown. Firmware writes values 03h, 01h, and VAR.
2-15  Always zero
```
Used in combination with Port 48081A0h, so it's probably power-related, too.

48081A4h - W_X_1A4h - (R/W) ffff [0000]
"Rate used when signal test is enabled (0x0A or 0x14 for 1 or 2 mbit)"
(Not too sure if that's correct, there is no visible relation to any rate.)
(This register seems to be R/W only on certain Port 48081A0h settings.)
Unknown. Firmware writes whatever.

4808290h - W_X_290h - (R/W or Disabled)
Reportedly, this is the "antenna" register, which should exist on official devkits, allowing to switch between wired Ethernet, and wireless Wifi mode.
```
0     Unknown (R/W) (if present)
1-15  Not used
```
On normal NDS release versions, this register seems to be disabled (if it is implemented at all), and trying to read from it acts as for unused registers, ie. reads return FFFFh (or probably 0000h on NDS-lite). The NDS firmware contains code for accessing this port, even in release versions.

W_INTERNAL
All registers marked as "W_INTERNAL" aren't used by Firmware part4, and are probably unimportant, except for whatever special diagnostics purposes.

Wifi DMA
Wifi RAM can be accessed with normal "Start Immediately" DMA transfers (typically by reading through W_RXBUF_RD_DATA, so the DMA automatically wraps from END to BEGIN).
Additionally, DMA0 and DMA2 can be reportedly synchronized to "Wireless Interrupt" (rather than using "Start Immediately" timing), no idea if/how that's working though... and if it gets started on any Wifi IRQ, or only on specific IRQs...?
Possibly some of the above unknown registers, or some unknown bits in other registers, are DMA related...?

Reportedly, early firmwares did use "Wireless Interrupt" DMAs (that'd be firmware v1/v2... or, only earlier unreleased prototype versions?).

DS Wifi Unused Registers

Wifi WS0 and WS1 Regions in NDS7 I/O Space
Wifi hardware occupies two 32K slots, but most of it is filled with unused or duplicated regions. The timings (waitstates) for WS0 and WS1 are initialized in WIFIWAITCNT (by firmware).

```
4800000h-4807FFFh Wifi WS0 Region (32K)
4808000h-4808000h Wifi WS1 Region (32K)
4810000h-4FFFFFFh Not used (00h-filled)
```

Structure of the 32K Wifi Regions (WS0 and WS1)

```
Wifi-WS0-Region    Wifi-WS1-Region    Content
4800000h-4800FFFh  4808000h-4808FFFh  Registers
4801000h-4801FFFh  4809000h-4809FFFh  Registers (mirror)
4802000h-4803FFFh  480A000h-480BFFFh  Unused
4804000h-4805FFFh  480C000h-480DFFFh  Wifi RAM  (8K)
4806000h-4806FFFh  480E000h-480EFFFh  Registers (mirror)
4807000h-4807FFFh  480F000h-480FFFFh  Registers (mirror)
```

Wifi Registers (recommended 4808000h-4808FFFh) appear more stable in WS1?
Wifi RAM (recommended 4804000h-4805FFFh) appears more stable in WS0?

Unused Ports (Original NDS)
Aside from those ports listed in the Wifi I/O Map, all other ports in range 4808000h..4808FFFh are unused. On the original DS, reading from these ports returns FFFFh.

Unused Ports (NDS-Lite)
Reading from unused I/O ports acts as PASSIVE mirror of W_RXBUF_RD_DATA. Exceptions are: Ports 4808188h, and 48082D8h..48082E6h; which always return 0000h.

Unused Memory (Original NDS)
Unused Wifi Memory is at 2000h..3FFFh. On the original DS, reading from that region returns FFFFh.

Unused Memory (NDS-Lite)
Reading from unused memory acts as PASSIVE mirror of WifiRAM (ie. reading from it returns the value being most recently read from 4000h..5FFFh) (that not affected by indirect WifiRAM reads via W_RXBUF_RD_DATA) (and, that not affected by writes to wifi memory, including writes that do overwrite the most recent read value) (and, that only if WifiRAM is properly enabled, ie. Port 220h.Bits0-1 should be 0).
Moreover, certain addresses are additionally ORed with mirrored I/O Ports. That addresses are:

```
2030h, 2044h, 2056h, 2080h, 2090h, 2094h, 2098h, 209Ch, 20A0h, 20A4h,
20A8h, 20AAh, 20B0h, 20B6h, 20BAh, 21C0h, 2208h, 2210h, 2244h, 31D0h,
31D2h, 31D4h, 31D6h, 31D8h, 31DAh, 31DCh, 31DEh.
```

For example, 2044h is a PASSIVE mirror of WifiRAM, ORed with an ACTIVE mirror of W_RANDOM (Port 044h). Note that some mirrors are at 2000h-2FFFh, and some at 3000h-3FFFh. The W_CMD_STAT mirrors are PASSIVE (that, in unused memory region only) (in normal port-mirror regions like 1000h-1FFF, W_CMD_STAT mirrors are ACTIVE).

Known (W) Mirrors (when reading from Write-only ports)

```
Read from (W)          Mirrors to (NDS)       Or to (NDS-Lite)
070h W_TXBUF_WR_DATA   060h W_RXBUF_RD_DATA   074h W_TXBUF_GAP
078h W_INTERNAL        068h W_TXBUF_WR_ADDR   074h W_TXBUF_GAP
0ACh W_TXREQ_RESET     09Ch W_INTERNAL        ? (zero)
0AEh W_TXREQ_SET       09Ch W_INTERNAL        ? (zero)
0B4h W_TXBUF_RESET     0B6h W_TXBUSY          ? (zero)
158h W_BB_CNT          15Ch W_BB_READ         ? (zero)
15Ah W_BB_WRITE        ? (zero)               ? (zero)
```

```
178h W_INTERNAL         17Ch W_RF_DATA2        ? (zero)
20Ch W_INTERNAL         09Ch W_INTERNAL        ? (zero)
21Ch W_IF_SET           010h W_IF              010h-OR-05Ch-OR-more?
228h W_X_228h           ? (zero)               ? (zero)
298h W_INTERNAL         084h W_TXBUF_TIM       084h W_TXBUF_TIM
2A8h W_INTERNAL         238h W_INTERNAL        238h W_INTERNAL
2B0h W_INTERNAL         084h W_TXBUF_TIM       084h W_TXBUF_TIM
```

Notes: The mirror to W_RXBUF_RD_DATA is a passive mirror.
The DS-Lite mirror at 21Ch consists of several ports ORed with each other (known components are Ports 010h and 05Ch, but there seem to be even more values ORed with it).

Port Mirror Regions
The Wifi Port region at 000h..FFFh is mirrored to 1000h..1FFFh, 6000h..6FFFh, and 7000h..7FFFh. Many of that mirrored ports are PASSIVE mirrors. Eg. reading from 1060h (mirror of Port 060h, W_RXBUF_RD_DATA) returns the old W_RXBUF_RD_DATA value (but without loading a new value from Wifi RAM, and without incrementing W_RXBUF_RD_ADDR). However, other registers, like W_RANDOM do have ACTIVE mirrors.

DS Wifi Initialization

Initialization sequence
These events must be done somewhat in sequence. There is some flexibility as to how they can be ordered but it's best to follow this order:

```
[4000304h].Bit1 = 1 ;POWCNT2  ;-Enable power to the wifi system
W_MACADDR = firmware[036h]    ;-Set 48bit Mac address
reg[012h] = 0000h   ;W_IE     ;-Disable interrupts
```

Wake Up the wireless system:

```
reg[036h] = 0000h ;W_POWER_US ;\clear all powerdown bits
delay 8 ms                    ; (works without that killer-delay ?)
reg[168h] = 0000h ;W_BB_POWER ;/
IF firmware[040h]=02h         ;\
  temp=BB[01h]                ; for wifitype=02h only:
  BB[01h]=temp AND 7Fh        ; reset BB[01h].Bit7, then restore old BB[01h]
  BB[01h]=temp                ; (that BB setting enables the RF9008 chip)
ENDIF                         ;/
delay 30 ms                   ;-(more killer-delay now getting REALLY slow)
call init_sub_functions       ;- same as "Init 16 registers by firmware[..]"
                              ;  and "Init RF registers", below.
                              ;  this or the other one probably not necessary
```

Init the Mac system:

```
reg[004h] = 0000h   - W_MODE_RST       ;set hardware mode
reg[008h] = 0000h   - W_TXSTATCNT      ;
reg[00Ah] = 0000h   - ? W_X_00Ah       ;(related to rx filter)
reg[012h] = 0000h   - W_IE             ;disable interrupts (again)
reg[010h] = FFFFh   - W_IF             ;acknowledge/clear any interrupts
reg[254h] = 0000h   - W_CONFIG_254h    ;
reg[0B4h] = FFFFh   - W_TXBUF_RESET    ;--reset all TXBUF_LOC's
reg[080h] = 0000h   - W_TXBUF_BEACON   ;disable automatic beacon transmission
reg[02Ah] = 0000h   - W_AID_FULL       ;\clear AID
reg[028h] = 0000h   - W_AID_LOW        ;/
reg[0E8h] = 0000h   - W_US_COUNTCNT    ;disable microsecond counter
reg[0EAh] = 0000h   - W_US_COMPARECNT  ;disable microsecond compare
reg[0EEh] = 0001h   - W_CMD_COUNTCNT   ;(is 0001h on reset anyways)
reg[0ECh] = 3F03h   - W_CONFIG_0ECh    ;
reg[1A2h] = 0001h   - ?                ;
reg[1A0h] = 0000h   - ?                ;
reg[110h] = 0800h   - W_PRE_BEACON     ;
reg[0BCh] = 0001h   - W_PREAMBLE       ;disable short preamble
reg[0D4h] = 0003h   - W_CONFIG_0D4h    ;
```

```
  reg[0D8h] = 0004h   - W_CONFIG_0D8h    ;
  reg[0DAh] = 0602h   - W_RX_LEN_CROP    ;
  reg[076h] = 0000h   - W_TXBUF_GAPDISP  ;disable gap/skip (offset=zero)
```
Init 16 registers by firmware[044h..063h]
```
  reg[146h] = firmware[044h] ;W_CONFIG_146h
  reg[148h] = firmware[046h] ;W_CONFIG_148h
  reg[14Ah] = firmware[048h] ;W_CONFIG_14Ah
  reg[14Ch] = firmware[04Ah] ;W_CONFIG_14Ch
  reg[120h] = firmware[04Ch] ;W_CONFIG_120h
  reg[122h] = firmware[04Eh] ;W_CONFIG_122h
  reg[154h] = firmware[050h] ;W_CONFIG_154h
  reg[144h] = firmware[052h] ;W_CONFIG_144h
  reg[130h] = firmware[054h] ;W_CONFIG_130h
  reg[132h] = firmware[056h] ;W_CONFIG_132h
  reg[140h] = firmware[058h] ;W_CONFIG_140h
  reg[142h] = firmware[05Ah] ;W_CONFIG_142h
  reg[038h] = firmware[05Ch] ;W_POWER_TX
  reg[124h] = firmware[05Eh] ;W_CONFIG_124h
  reg[128h] = firmware[060h] ;W_CONFIG_128h
  reg[150h] = firmware[062h] ;W_CONFIG_150h
```
Init RF registers
```
  numbits = BYTE firmware[041h]    ;usually 18h
  numbytes = (numbits+7)/8         ;usually 3
  reg[0x184] = (numbits+80h) AND 017Fh  -- W_RF_CNT
  for i=0 to BYTE firmware[042h]-1 ;number of entries (usually 0Ch) (0..0Bh)
   if BYTE firmware[040h]=3
    RF[i]=firmware[0CEh+i]
   else
    RF_Write(numbytes at firmware[0CEh+i*numbytes])
   endif
```
Init the BaseBand System
```
  (this should be not required, already set by firmware bootcode)
  reg[160h] = 0100h  ;W_BB_MODE
  BB[0..68h] = firmware[64h+(0..68h)]
```
Set Mac address
```
  copy 6 bytes from firmware[036h] to mac address at 0x04800018  (why again ?)
```
Now just set some default variables
```
  reg[02Ch]=0007h  ;W_TX_RETRYLIMIT - XXX needs to be set for every transmit?
  Set channel (see section on changing channels)
  Set Mode 2 -- sets bottom 3 bits of W_MODE_WEP to 2
  Set Wep Mode / key -- Wep mode is bits 3..5 of W_MODE_WEP
  BB[13h] = 00h  ;CCA operation (use only carrier sense, without ED)
  BB[35h] = 1Fh  ;Energy Detection Threshold (ED)
```
-- To further init wifi to the point that you can properly send
-- and receive data, there are some more variables that need to be set.
```
  reg[032h] = 8000h -- W_WEP_CNT     ;Enable WEP processing
  reg[134h] = FFFFh -- W_BEACONCOUNT2;reset post-beacon counter to LONG time
  reg[028h] = 0000h -- W_AID_LOW     ;\clear W_AID value, again?!
  reg[02Ah] = 0000h -- W_AID_FULL    ;/
  reg[0E8h] = 0001h -- W_US_COUNTCNT ;enable microsecond counter
  reg[038h] = 0000h -- W_POWER_TX    ;disable transmit power save
  reg[020h] = 0000h -- W_BSSID_0     ;\
  reg[022h] = 0000h -- W_BSSID_1     ; clear BSSID
  reg[024h] = 0000h -- W_BSSID_2     ;/
```
-- TX prepare
```
  reg[0AEh] = 000Dh -- W_TXREQ_SET   ;flush all pending transmits (uh?)
```
-- RX prepare
```
  reg[030h] = 8000h    W_RXCNT         ;enable RX system (done again below)
  reg[050h] = 4C00h    W_RXBUF_BEGIN   ;(example values)
  reg[052h] = 5F60h    W_RXBUF_END     ;(length = 4960 bytes)
  reg[056h] = 0C00h/2  W_RXBUF_WR_ADDR ;fifo begin latch address
  reg[05Ah] = 0C00h/2  W_RXBUF_READCSR     ;fifo end, same as begin at start.
```

```
  reg[062h] = 5F60h-2  W_RXBUF_GAP     ;(set gap<end) (zero should work, too)
  reg[030h] = 8001h    W_RXCNT  ;enable, and latch new fifo values to hardware
--
  reg[030h] = 8000h    W_RXCNT       enable receive (again?)
  reg[010h] = FFFFh    W_IF          clear interrupt flags
  reg[012h] = whatever W_IE          set enabled interrupts
  reg[1AEh] = 1FFFh    W_RXSTAT_OVF_IE desired STAT Overflow interrupts
  reg[1AAh] = 0000h    W_RXSTAT_INC_IE desired STAT Increase interrupts
  reg[0D0h] = 0181h    W_RXFILTER set to 0x581 when you successfully connect
                        to an access point and fill W_BSSID with a mac
                        address for it. (W_RXFILTER) [not sure on the values
                        for this yet]
  reg[0E0h] = 000Bh  -- W_RXFILTER2     ;
  reg[008h] = 0000h  -- ? W_TXSTATCNT   ;(again?)
  reg[00Ah] = 0000h  -- ? W_X_00Ah      ;(related to rx filter) (again?)
  reg[004h] = 0001h  -- W_MODE_RST      ;hardware mode
  reg[0E8h] = 0001h  -- W_US_COUNTCNT   ;enable microsecond counter (again?)
  reg[0EAh] = 0001h  -- W_US_COMPARECNT ;enable microsecond compare
  reg[048h] = 0000h  -- W_POWER_?    ;[disabling a power saving technique]
  reg[038h].Bit1 = 0 -- W_POWER_TX   ;[this too]
  reg[048h] = 0000h  -- W_POWER_?    ;[umm, it's done again. necessary?]
  reg[0AEh] = 0002h  -- W_TXREQ_SET  ;
  reg[03Ch].Bit1 = 1 -- W_POWERSTATE ;queue enable power (RX power, we believe)
  reg[0ACh] = FFFFh  -- W_TXREQ_RESET;reset LOC1..3
```

That's it, the DS should be now happy to send and receive packets.
It's very possible that there are some unnecessary registers set in here.

DS Wifi Flowcharts

Wifi Transmit Procedure

To transmit data via wifi (Assuming you've already initialized wifi and changed channels to the channel you want):

```
 (1) Copy the TX Header followed by the 802.11 packet to send anywhere it
      will fit in MAC memory (halfword-aligned)
 (2) Take the offset from start of MAC memory that you put the packet,
      divide it by 2, and or with 0x8000 - store this in one of the
      W_TXBUF_LOC registers
 (3) Set W_TX_RETRYLIMIT, to allow your packet to be retried until an ack is
      received (set it to 7, or something similar)
 (4) Store the bit associated with the W_TXBUF_LOC register you used
      into W_TXREQ_SET - this will send the packet.
 (5) You can then read the result data in W_TXSTAT when the TX is over
      (you can tell either by polling or interrupt) to find out how many
      retries were used, and if the packet was ACK'd
```

Of course, this is just the simplest approach, you can be a lot more clever about it.

Wifi Receive Procedure

To receive data via wifi, you either need to handle the wifi received data interrupt, or you need to poll W_RXBUF_WRCSR - whenever it is != W_RXBUF_READCSR, there is a new packet. When there is a new packet, take the following approach:

```
 (1) Calculate the length of the new packet (read "received frame length"
      which is +8 bytes from the start of the packet) - total frame length
      is (12 + received frame length) padded to a multiple of 4 bytes.
 (2) Read the data out of the RX FIFO area (keep in mind it's a circular
      buffer and you may have to wrap around the end of the buffer)
 (3) Set the value of W_RXBUF_READCSR to the location of the next packet
      (add the length of the packet, and wrap around if necessary)
```

Keep in mind, W_RXBUF_READCSR and W_RXBUF_WRCSR must be multiplied by 2 to get a byte offset from the start of MAC memory.

Wifi Change Channels Procedure (ch=1..14)
For Type2 or Type5 (ie. firmware[040h]<>3): ;(Type2, used in Original-DS)
```
RF[firmware[F2h+(ch-1)*6]/40000h] = firmware[F2h+(ch-1)*6] AND 3FFFFh
RF[firmware[F5h+(ch-1)*6]/40000h] = firmware[F5h+(ch-1)*6] AND 3FFFFh
delay a few milliseconds  ;huh?
IF RF[09h].bit16=0     ;External Gain (default)
 BB[1Eh]=firmware[146h+(ch-1)]                          ;set BB.Gain register
ELSEIF RF[09h].bit15=0 ;Internal Gain from TXVGC Bits
 RF[09h].Bit10..14 = (firmware[154h+(ch-1)] AND 1Fh)   ;set RF.TXVGC Bits
ENDIF
```
For Type3 (ie. firmware[040h]=3): ;(Type3, used in DS-Lite)
```
num_initial_regs = firmware[042h]
addr=0CEh+num_initial_regs
num_bb_writes = firmware[addr]
num_rf_writes = firmware[43h]
addr=addr+1
for i=1 to num_bb_writes
  BB[firmware[addr]] = firmware[addr+ch]
  addr=addr+15
next i
for i=1 to num_rf_writes
  RF[firmware[addr]] = firmware[addr+ch]
  addr=addr+15
next i
```
Congrats, you are now ready to transmit/receive on whatever channel you picked.

Channels
The IEEE802.11b standard (and the NDS hardware) support 14 channels (1..14).
Channels 1..13 use frequencies 2412MHz..2472MHz (in 5MHz steps). Channel 14 uses frequency 2484MHz.
Which channels are allowed to be used varies from country to country, as indicated by Bit1..14 of firmware[03Ch]. Channel 14 is rarely used (dated back to an older japanese standard).

Caution: Nearby channels do overlap, you'll get transmission errors on packets that are transferred simultaneously with packets on nearby channels. But, you won't successfully receive packets from nearby channels (so you won't even "see" that they are there, which is bad, as it doesn't allow you to share the channel synchronized with other hosts; ie. it'd be better if two hosts are using the SAME channel, rather than to use nearby channels).
To avoid that problem, conventionally only channels 1,6,11 are used - however Nintendo uses channels 1,7,13 - which is causing conflicts between channel 6,7, and maybe also between 11,13.

DS Wifi Hardware Headers

Hardware TX Header (12 bytes) (TXHDR)
The TX header immediately precedes the data to be sent, and should be put at the location that will be given to the register activating a transmission.
```
Addr Siz Expl.
00h  2   Status - In: Don't care - Out: Status (0000h=Failed, 0001h=Okay)
02h  2   Unknown - In: Don't care
           Bit0: Usually zero.
           Bit1..15 --------> flags for multiboot slaves number 1..15
           (Should be usually zero, except when sending multiplay commands
           via W_TXBUF_CMD. In that case, the slave flags should be ALSO
           stored in the second halfword of the FRAME BODY. Actually, the
           hardware seems to use only that entry (in the BODY), rather than
           using this entry (in the hardware header)).
04h  1   Unknown - In: Must be 00h..02h (should be 00h)
           (03h..FFh result in error: W_TXSTAT.Bit1 gets set, but
```

```
            nethertheless header entry[00h] is kept set to 0001h=Okay)
            ;00h = use W_TX_SEQNO (if enabled in TXBUF_LOCn)
            ;01h = force NOT to use W_TX_SEQNO (even if it is enabled in LOCn)
            ;02h = seems to behave same as 01h
  05h  1   Unknown - In: Don't care - Out: Set to 00h
  06h  2   Unknown - In: Don't care
  08h  1   Transfer Rate (0Ah=1Mbit/s, 14h=2Mbit/s) (other values=1MBit/s, too)
  09h  1   Unknown - In: Don't care
  0Ah  2   Length of IEEE Frame Header+Body+checksum(s) in bytes
           (14bits, upper 2bits are unused/don't care)
```

The eight "Don't care" bytes should be usually set to zero (although setting them to FFh seems to be working as well). Entries [00h] and [05h] are modified by hardware, all other entries are kept unchanged.

Important note! TX length includes the length of a 4-byte "FCS" (checksum) for the packet. The hardware generates the FCS for you, but you still must include it in the packet length. Also note that if the 802.11 WEP enabled bit is set in the header, the packet will be automatically encrypted via the wep algorithm - however, the software is responsible for providing the 4-byte IV block with the WEP key ID and the 24bit IV value. - ALSO, you must include the length of the *encrypted* FCS used in packets that have wep enabled (increase the tx length by another 4 bytes) - this value is calculated automaticly for you, but you are responsible for including it in the length of your packet (if you have data there, it'll be replaced by the FCS.)

Hardware RX Header (12 bytes) (RXHDR)

The RX header is an informational structure that provides needed information about a received packet. It is written right before the received packet data in the rx circular buffer.

```
  Addr Siz Expl.
  00h  2   Flags
             Bit0-3: Frame type/subtype:
               0  managment/any frame (except beacon and invalid subtypes)
               1  managment/beacon frame
               5  control/ps-poll frame
               8  data/any frame (subtype0..7) (ie. except invalid subtypes)
               C,D,E,F  unknown (firmware is checking for that values)
               ---
               C    firmware uses it for data/cf-poll frame, FromDs (*)
               D    firmware uses it for data/cf-ack frame, FromDs
               E,F  firmware uses it for data/cf-ack frame, ToDs
               (*) with DA=broadcast
               ---
             Bit4:   Seems to be always set
             Bit5-7: Seems to be always zero
             Bit8: Set when FC.Bit10 is set (more fragments)
             Bit9: Set when the lower-4bit of Sequence Control are nonzero,
                   it is also set when FC.Bit10 is set (more fragments)
                   So, probably, it is set on fragment-mismatch-errors
             Bit10-14: Seems to be always zero
             Bit15: Set when Frame Header's BSSID value equals W_BSSID register
  02h  2   Unknown (usually 0040h)
  04h  2   Time since last packet (eg. when receiving beacons: total random on
            first some packets, but later on it gets equal to Beacon Interval)
            In other cases, this value is equal to the 1st 2 bytes of the DA ?
            [Above time/da effects might be explained by other reason: maybe
            this entry is left unchanged, simply containing old WifiRAM value?]
  06h  2   Transfer Rate (N*100kbit/s) (ie. 14h for 2Mbit/s)
  08h  2   Length of IEEE Frame Header+Body in bytes (excluding FCS checksum)
  0Ah  1   MAX RSSI   ;\Received Signal Strength Indicator
  0Bh  1   MIN RSSI   ;/
```

Important Note: Received frame lengths are always multiples of 4 bytes. While the actual header length + received frame length may be less, when incrementing the read cursor you must pad the length to a multiple of 4 bytes.

IEEE Header

The above Hardware headers should (must) be followed by valid IEEE headers. Although that headers are to be generated by software, the hardware does do some interaction with the IEEE headers, such like comparing address fields with W_MACADDR and W_BSSID. And, it does modify some entries of it:
1) The sequence control value is replaced by W_TX_SEQNO*10h (when enabled in W_TXBUF_LOCn.Bit13), this replacement does also overwrite the local TXBUF value.
2) The frame control value is modified, namely, the hardware tends to set Bit12 of it. This replacement does NOT modify the local TXBUF, but the remote RXBUF will receive the modified value. Also, Bit0-1 (protocol version) are forcefully set to 0.
3) Transmits via W_TXBUF_BEACON do additionally modify the 64bit timestamp (so W_TXBUF_BEACON should be used ONLY for packets WITH timestamp, ie. Beacons or Probe-Responses). The local TXBUF seems to be left unchanged, but the remote RXBUF will contain the (sender's) W_US_COUNT value.
C) For Control Frames, the hardware headers Length value is transferred as normally (ie. excluding the FCS length, remote RXBUF will contain TXBUF length minus 4), but - no matter of that length value - only 10 or 16 bytes (depending on the subtype) of the IEEE frame are actually transferred and/or stored in RXBUF.
X) For Control Frames with Subtype 0Ah, the AID entry is set to C000h, that, probably ORed with original value in WifiRAM, or with the W_AID_FULL register?
XX) No idea if it's possible to send Control Frames with subtype 0Bh..0Fh, as for now, it seems that either they aren't sent, or the receipient is ignoring them (or processing them internally, but without storing them in RXBUF).

DS Wifi Nintendo Beacons

Available Game Advertisement
WMB uses beacon frames to advertise available games for download. The beacon frames are normally used to advertise available access points in most 802.11 systems, but there is nothing preventing their use in this capacity. The advertisement data is fragmented and stored partially in each beacon frame as the payload of a custom information element (tag: 0xDD).

The DS Download Play menu only lists games when the beacons are broadcasted on one of the following channels: 1, 3, 4, 5, 7, 9, 10, 11, 13, and 14 (that is WRONG, firmware_v3 checks only channels 1,7,13). However, the DS hosting mechanism only seems to transmit on channels 1, 7, and 13 (apparently selected at random).

All beacon frames transmitted by a DS host have the following format

```
802.11 management frame
802.11 beacon header (Timestamp,BeaconInterval,Capability?)
Supported rates (tagged IE, advertises 1 Mbit and 2 Mbit)
DS parameter set (tagged IE, note: Distribution System, not Nintendo DS)
TIM vector (tagged IE, transmitted as empty)
Custom extension (tagged IE, tag DDh, see below)
```

Nintendo specific beacon fragment format (information element code DDh)

```
Offset Description
00h  Nintendo Beacon OUI (00h,09h,BFh,00h)
04h  Stepping Offset for 4808134h/W_BEACONCOUNT2 (always 000Ah)
06h  Strange Timestamp (W_US_COUNT*2-VCOUNT*7Fh)/128 (0000h for multiboot)
08h  01 00
0Ah  40 00
0Ch  24 00
0Eh  40 00
10h  Randomly generated stream code
12h  Number of bytes from entry 18h and up (70h for multiboot) (0 if Empty)
13h  Beacon Type    (0Bh=Multiboot, 01h=Multicart/Pictochat, 09h=Empty)
14h  0100 0008    (some kind of max,min values?)
```

For Empty (length zero, is used at very begin of multiboot)

```
18h  No data.
```

For Multicart (variable length)

```
  18h  Custom data, usually containing the host name, either in 8bit ascii,
       or 16bit unicode format. Sometimes taken from Firmware User Settings,
       and sometimes from Cartridge Backup Memory.
```

For Pictochat (length 8)

```
  18h  Fixed (always 2348h)
  1Ah  xxxx
  1Ch  Chatroom number (00h..03h for Chatroom A..D)
  1Dh  Number of users already connected (01h..10h) (including host)
  1Eh  Fixed (always 0004h)
```

For Multiboot (always 70h bytes)

```
  18h  24 00 40 00 (varies from game to game)
  1Ch  End of advertisement flag (00 for non-end, 02 for end packets)
  1Dh  Always 00, 01, 02, or 04
  1Eh  Number of players already connected
  1Fh  Sequence number (0 .. total_advertisement_length)
  20h  Checksum (on entries 22h and up)
         chksum=0, for i=22h to 86h step 2, chksum=chksum+halfword[i], next i,
         chksum=FFFFh AND NOT (chksum+chksum/10000h)
  22h  Sequence number in non-final packet, # of players in final packet
  23h  Total advertisement length - 1 (in beacons)
  24h  Datasize in bytes (2 byte little-endian)
          (0062h for seq 0..7, 0048h for seq 8, 0001h for seq 9)
  26h  Data (always 62h bytes, padded with 00h if Datasize<62h)
```

Multiboot Advertisement Fragments

The advertisement fragments are reordered and assembled according to their internal sequence number, to form the overall advertisement payload, as defined below:

```
  Offset Size Description
  000h  32  Icon Palette (same as for ROM Cartridge Icon)
  020h  512 Icon Bitmap  (same as for ROM Cartridge Icon)
  220h  1   Unknown (0Bh)
  221h  1   Length of hosting name          ;(probably same as firmware
  222h  20  Name of hosting DS (10 UCS-2)   ;user name?)
  236h  1   Max number of players
  237h  1   Unknown (00h)
  238h  96  Game name (48 UCS-2)   (same as 1st line of ROM Cartridge Title)
  298h  192 Description (96 UCS-2) (same as further lines of ROM Cart Title)
  358h  64  00's if no users are connected  <---WRONG: LEN=1, not 64
  398h  0   End of data if no users are connected
```

Authentication process

Once a user B chooses a download offered by a host A, the following standard 802.11 authentication process observed.

```
  Host A advertises a game in beacon frames as described above
  Client B sends an authentication request (sequence 1) to A    ;\step 1
  Host A replies with an ACK                                    ;/
  Host A sends an authentication reply (sequence 2) to B        ;\step 2
  (uh, no ACK here?)                                            ;/
  Client B sends an association request                         ;\step 3
  Host A replies with an ACK                                    ;/
  Host A sends an association response                          ;\step 4
  Client B replies with an ACK                                  ;/
```

After this, the two are associated, and will remain so until the transfer is complete or one is idle for several seconds, at which point they will de-associate. For more information on the association process, see the 802.11 standard.

DS Wifi Nintendo DS Download Play

Before downloading: Receive beacons, display Icon/Title, prompt user whether to download that title, and, if so, connect to host (via authentication and association requests). See previous chapter for details.

Download process (after authentication)

```
  Host sends some Pings, client(s) send PongReplies
  Host sends more Pings, client(s) send UsernameReplies
  Host sends RSA frame, client(s) send RsaReply
  Host sends NDS header, client(s) send DataReply
  Host sends ARM9 binary, client(s) send DataReply
  Host sends ARM7 binary, client(s) send DataReply
  Host sends Done message, no reply from client(s)
```

The WMB protocol implements layers 3 to 7 of the OSI network model, but does not define a new type of network addresses. However, it does define a couple of special broadcast-like MAC addresses within the assigned Nintendo namespace (00:09:BF:xx:xx:xx).

Flows

The three channels or "flows" used for all communications after the MAC broadcast beacons take the form 03:09:BF:00:00:xx:

```
  03:09:BF:00:00:00  host to client main data flow ;via 4808090h/W_TXBUF_CMD
  03:09:BF:00:00:10  client to host replies        ;via 4808094h/W_TXBUF_REPLY1
  03:09:BF:00:00:03  host to client feedback flow  ;acknowledges the replies?
```

The host does something unusual with the 802.11 sequence control field, each packet sent out on the 03:09:BF:00:00:00 flow has a sequence control number 2 greater than the previous one, even if they are sent sequentially. When the host acknowledges a reply (on flow 03:09:BF:00:00:03) from the client about a particular packet, it uses the sequence number one after the original packet number it sent out on 00. This is the root of one of the major problems in finding a PC card that can transmit WMB packets, as very few cards provide user control over it. Even when a card is capable of 'raw' 802.11 transmission, it typically takes care of the sequence control field in hardware or firmware, filling it with a constantly incrementing number.

Host-to-client packets (on the 03:09:BF:00:00:00 flow)

Sent via via 4808090h/W_TXBUF_CMD (multiplayer master)

```
  00h 2   Value for W_CMD_REPLYTIME (0106h)
  02h 2   Slave Flags, bit1..15 for slave 1..15 (1=connected) (eg. 0002h)
  04h 1   Size in halfwords
  05h 1   Flags
  06h 1   Command (01h=Ping/NameRequest, 03h=RSA, 04h=DataPacket, 05h=Done)
 For Command 01h (Ping):                              ;\
  07h 4   Unused (zerofilled)                         ;
 For Command 03h (RSA):                               ;
  07h E0h RSA Signature Frame (see below)             ; Payload
 For Command 04h (Data Packet):                       ;
  07h 1   Unknown (zero)                              ;
  08h 2   Packet Number (0000h and up)                ;
  0Ah ..  Data                                        ;
 For Command 05h (Done):                              ;
  07h ..  Unknown                                     ;/
 For all commands (whatever trailing three bytes):
  xxh 3   Unknown (00h,02h,00h)
```

The size field is in terms of half-words (16 bits), and includes the flags byte along with the payload (so a size of 0x03 represents a flag byte, a command byte, and 4 bytes of payload).

When flags is 0x11, the first byte of the payload is a command. There seems to be no important data when flags is not 0x11 (seen occasionally as 0x01), and ignoring them still results in a complete dump.

Data packets (type 0x04) include a transport-layer sequence number inside of the data packet itself, but no destination offset or other mechanism to allow the packets to be processed out-of-order. The only way to place the data at the correct location in memory is to re-order the packets according to the sequence number and process them sequentially. The sequence number is a zero based little-endian number. Each packet only contains

data for one of the three destination blocks (header, ARM9, ARM7), so the change-of-destination check only needs to be made on packet boundaries.

Client to Host Replies (on the 03:09:BF:00:00:10 flow)
Sent via 4808094h/W_TXBUF_REPLY1 (multiplayer slave)
The client uses an incrementing sequence control number for all of its packets, with no unusual trickery. Each reply is sent as a standard 802.11 data frame (typically as a Data + CF-Acknowledgement), consisting of ten data bytes:

```
  00h 2  Unknown/fixed (8104h)
  02h 1  Reply Type (00h=Pong, 07h=Username, 08h=RsaReply, 09h=DataReply)
 For Reply Type 00h (PongReply):
  03h 7  Unused (zerofilled)
 For Reply Type 07h (UsernameReply):
  03h 1  Username fragment (01h..04h)
  04h 6  Username Char[0,1,2]                    ;for fragment 01h
  04h 6  Username Char[3,4,5]                    ;for fragment 02h
  04h 6  Username Char[6,7,8]                    ;for fragment 03h
  04h 6  Username Char[9], bytes 01h,00h,00h,00h ;for fragment 04h
 For Reply Type 08h (RsaReply):
  03h 7  Unused (garbage, usually same as Name fragment 02h)
 For Reply Type 09h (DataReply):
  03h 2  Last packet (the packet number being acknowledged)
  05h 2  Best packet (the highest continuous packet number seen so far)
  07h 3  Unused (zerofilled)
```

The NameReply is sent shortly after association is completed, although I am not certain what triggers it. There are a variable number of Pings preceding this reply, but most are replied via Pongs. The name reply sends the user-configured DS name (set in the firmware menu) split over four messages (with the 4th byte of the packet specifying which message fragment this is, 1 based). This can be a total length of 10 UCS-2 characters, although all four messages are still sent if it is shorter (padded with nulls to 10 characters, and then 01 and then nulls until the end of the frame).

Host to client acknowledgements (on the 03:09:BF:00:00:03 "feedback" flow)
Sent via UNKNOWN which port sends this?
Maybe "second half" of W_TXBUF_CMD? Or hardware auto-sends this as "ack"?

```
  00h 1  Unknown (random/garbage?) (unknown value... maybe per slave flags?)
  01h 3  Unused (zerofilled)
```

These packets contain four data bytes, but three are always zero. The first seems to be random, with no connection to the acknowledged data. The actual indication of acknowledgement is the sequence control number of the packet. It is set to be one greater than the sequence control number of the initial host packet (sent on flow 03:09:BF:00:00:00) that the client has just responded to, to indicate that the reply was received.

RSA signature frame payload (type 0x03)
The RSA frame format (type 0x03) sends a table of information about the game being downloaded (most of it redundant with the NDS header), as well as the RSA signature for the DS.
Notably: the RSA frame itself is not included as part of the data being signed, bringing up various security issues and making Nintendo's firmware engineers look amateurish at best.
There are several abortive sendings of empty RSA frames with a size field of 0x03, before the real frame is sent (always with a size field of 0x75).

```
  00h 4   ARM9 execute address
  04h 4   ARM7 execute address
  08h 4   Zerofilled
  0Ch 4   Header destination (temp)
  10h 4   Header destination (actual)
  14h 4   Header size (160h)
  18h 4   Zerofilled
  1Ch 4   ARM9 destination address (temp)
  20h 4   ARM9 destination address (actual)
  24h 4   ARM9 binary size
  28h 4   Zerofilled
```

```
  2Ch 4   ARM7 destination address (temp) (usually 22C0000h in Main RAM)
  30h 4   ARM7 destination address (actual) (usually somewhere in WRAM)
  34h 4   ARM7 binary size
  38h 4   Unknown (00000001h)
  3Ch 4   Signature ID (61h,63h,01h,00h) (aka "ac", or backwards "ca") ;\
  40h 80h Signature RSA (RSA signature in OpenPGP SHA1 format)         ;
  C0h 4   Signature Footer                                            ;/
  C4h 36  Zerofilled
  E8h -   End of frame payload
```

The offsets in the table are from after the command byte, ie. two bytes into the 234 bytes of payload including the flags.
The temp/actual destination addresses are usually identical, except for ARM7 which is loaded to a separate temp address in Main RAM (needed there for computing its SHA1 on ARM9 side).
The RSA signature is as so: First compute SHA1's on Header and ARM9 and ARM7 areas, and store them in a 40h-byte array:

```
  00h 14h SHA1 on Header
  14h 14h SHA1 on ARM9 bootcode
  28h 14h SHA1 on ARM7 bootcode
  3Ch 4   Signature Footer (the four bytes from [C0h])
```

Then compute a SHA1 on that 40h-byte array, that SHA1 value is then used in the RSA signature (in OpenPGP SHA1 format, ie. same as SWI 23h in DSi BIOS). The RSA private key is unknown, the RSA public key (9Eh,C1h,CCh,C0h,...) can be found in the NDS Firmware and in the DSi's DS Download Play utility.
Unknown if the 4-byte Footer value is having any special meaning (eg. it might be some checksum, or it might be just some random/timestamp).
The 4+80h+4 signature bytes at [3Ch..C3h] are conventionally stored at the "end" of the binary, ie. at the offset specified in cartheader[080h].

NDS format

The .NDS format is the standard format for Nintendo DS programs; it originated on original game cards and also appears to a limited extent in WMB binaries. The WMB process only transfers the first 0x160 bytes of the header, the ARM9 binary, and the ARM7 binary (in that order), ignoring the file name and file allocation tables, the overlay data, and some information stored in the banner (the rest is transmitted partially via the beacon advertisement process).

List of DS Download Play Games

Games with DS Download Play Support (single-cart multiplayer):

```
  Over the Hedge (download contains a 2D minigame)
```

Demos available from DS Download Stations:

```
  Eragon
  Lara Croft Tomb Raider Legend
  Magnetica
  Metroid Prime Hunters Demo
  Submarine Tech Demo
  (and many trailers with non-playable movie clips)
```

DS Wifi IEEE802.11 Frames

MAC Frame Format

```
  10..30 bytes    MAC Header
  0..2312 bytes   Frame Body (in practice, network MTU is circa 1500 bytes max)
  4 bytes         Frame Check Sequence (FCS) (aka checksum)
```

MAC Header (10..30 bytes)

```
  Size Content
  2    Frame Control Field (FC)
  2    Duration/ID
  6    Address 1
```

```
(6)   Address 2 (if any)
(6)   Address 3 (if any)
(2)   Sequence Control (if any)
(6)   Address 4 (if any)
```

Frame Control Field (FC)

```
Bit  Expl.
0-1  Protocol Version   (0=Current, 1..3=Reserved)
2-3  Type               (0=Managment, 1=Control, 2=Data, 3=Reserved)
4-7  Subtype            (see next chapters) (meaning depends on above Type)
8    To Distribution System (DS)
9    From Distribution System (DS)
10   More Fragments
11   Retry
12   Power Managment    (0=Active, 1=STA will enter Power-Safe mode after..)
13   More Data
14   Wired Equivalent Privacy (WEP) Encryption (0=No, 1=Yes)
15   Order
```

Bit 8-11 and Bit 13-15 are always 0 in Control Frames.

Duration/ID Field (16bit)

```
0000h..7FFFh  Duration (0-32767)
8000h         Fixed value within frames transmitted during the CFP
              (CFP=Contention Free Period)
8001h..BFFFh  Reserved
C000h         Reserved
C001h..C7D7h  Association ID (AID) (1..2007) in PS-Poll frames
C7D8h..FFFFh  Reserved
```

48bit MAC Addresses

MAC Addresses are 48bit (6 bytes) (Bit0 is the LSB of the 1st byte),

```
0     Group Flag (0=Individual Address, 1=Group Address)
1     Local Flag (0=Universally Administered Address, 1=Locally Administered)
2-23  22bit Manufacturer ID (assigned by IEEE)
24-47 24bit Device ID (assigned by the Manufacturer)
```

Special NDS related Addresses:

```
00 09 BF xx xx xx  NDS-Consoles (Original NDS with firmware v1-v5)
00 16 56 xx xx xx  NDS-Consoles (Newer NDS-Lite with firmware v6 and up)
00 23 CC xx xx xx  DSi-Consoles (Original DSi with early mainboard; nocash)
00 24 1E xx xx xx  DSi-Consoles (Another DSi; scanlime)
40 F4 07 xx xx xx  DSi-Consoles (with DWM-W024; nocash)
E0 E7 51 xx xx xx  DSi-Consoles (with DWM-W024; nocash/desoldered)
CC 9E 00 xx xx xx  DSi-Consoles (with J27H020; nocash)
03 09 BF 00 00 00  NDS-Multiboot: host to client (main data flow)
03 09 BF 00 00 10  NDS-Multiboot: client to host (replies)
03 09 BF 00 00 03  NDS-Multiboot: host to client (acknowledges replies)
FF FF FF FF FF FF  Broadcast to all stations (eg. Beacons)
```

Sequence Control Field

```
Bit  Expl.
0-3  Fragment Number (0=First (or only) fragment)
4-15 Sequence Number
```

(increment by 1, except on retransmissions, ie. retries)

WEP Frame Body

```
3 bytes     Initialization Vector (WEP IV)
1 byte      Pad (6bit, all zero), Key ID (2bit)
1..? bytes  Data (encrypted data)
4 bytes     ICV (encrypted CRC32 across Data)
```

DS Wifi IEEE802.11 Managment Frames (Type=0)

All Managment Frames have 24-byte Frame Header, with following values:

```
FC(2), Duration(2), DA(6), SA(6), BSSID(6), Sequence Control(2)
```

The content of the Frame Body depends on the FC's Subtype:

```
Subtype                   Frame Body
0 Association request     Capability, ListenInterval, SSID, SuppRates
1 Association response    Capability, Status, AID, SuppRates
2 Reassociation request   Capability, ListenInterval, CurrAP, SSID, SuppRates
3 Reassociation response  Capability, Status, AID, SuppRates
4 Probe request           SSID, SuppRates
5 Probe response          Same as for Beacon (but without TIM)
8 Beacon                  Timestamp,BeaconInterval,Capability,SSID,SuppRates,
                           FH Parameter Set (when using Frequency Hopping),
                           DS Parameter Set (when using Direct Sequence),
                           CF Parameter Set (when supporting PCF),
                           IBSS Parameter Set (when in an IBSS),
                           TIM (when generated by AP)
9 Announcement traffic indication message (ATIM)    Body is "null" (=none?)
A Disassociation          ReasonCode
B Authentication          AuthAlgorithm, AuthSequence, Status, ChallengeText
C Deauthentication        ReasonCode
```

Subtypes 6..7, and D..F are Reserved.

The separate components of the Frame Body are...

64bit Parameters (8 bytes)

```
Timestamp: value of the TSFTIMER (see 11.1) of a frame's source. Uh?
```

48bit Parameters (6 bytes)

```
Current AP (Access Point): MAC Address of AP with which station is associated
```

16bit Parameters (2 bytes)

```
Capability Information (see list below)
Status code (see list below) (0000h=Successful, other=Error code)
Reason code (see list below) (Error code)
Association ID (AID) (C000h+1..2007)
Authentication Algorithm (0=Open System, 1=Shared Key, 2..FFFFh=Reserved)
Authentication Transaction Sequence Number (Open System:1-2, Shared Key:1-4)
Beacon Interval (Time between beacons, N*1024 us)
Listen Interval (see note below)
```

Information elements (1byte ID, 1byte LEN, followed by LEN byte(s) data)

```
ID      LEN      Expl.
00h     00h-20h  SSID Service Set Identity (LEN=0 for broadcast SSID) (ASCII)
01h     01h-08h  Supported rates; each (nn AND 7Fh)*500kbit/s, bit7=flag
02h     05h      FH (Frequency Hopping) Parameter Set
                   DwellTime(16bit), HopSet, HopPattern, HopIndex
03h     01h      DS (Distribution System) Parameter Set; Channel (01h..0Eh)
04h     06h      CF Parameter Set; Count, Period, MaxDuration, RemainDuration
05h     04h..FEh TIM; Count,Period,Control, 1-251 bytes PartialVirtualBitmap
06h     02h      IBSS Parameter Set; ATIM Window length (16bit)
07h-0Fh -        Reserved
(07h)   ..       802.11d Country
(08h)   ..       802.11d Hopping Pattern Params
(09h)   ..       802.11d Hopping Pattern Table
(0Ah)   ..       802.11d Request
10h     02h..FEh Challenge text; 1-253 bytes Authentication data
                  (Used only for Shared Key sequence no 2,3)
                  (none such for Open System)
                  (none such for Shared key sequence no 1,4)
11h-1Fh -        Reserved for challenge text extension
20h-FFh -        Reserved
(20h)   ..       802.11h Power Constraint
(21h)   ..       802.11h Power Capability
(22h)   ..       802.11h TPC Request (Transmit Power Control)
```

```
(23h)    ..       802.11h TPC Report
(24h)    ..       802.11h Supported Channels
(25h)    ..       802.11h Channel Switch Announcement
(26h)    ..       802.11h Measurement Request
(27h)    ..       802.11h Measurement Report
(28h)    ..       802.11h Quiet
(29h)    ..       802.11h IBSS DFS
 2Ah     ..       802.11g ERP Information (spotted in newer beacons)
 30h     var      802.11i Reserved but used for WPA2 RSNIE <-- officially
 32h     ..       802.11g Extended Supported Rates (spotted in newer beacons)
 DDh     var      Reserved but used for WPA RSNIE  <-- vendor specific
 DDh     var      Reserved but used by Nintendo for NDS-Multiboot beacons
 2Dh     ..       Unknown (spotted in newer beacons)
 2Fh     ..       Unknown (spotted in newer beacons)
 3Dh     ..       Unknown (spotted in newer beacons)
 7Fh     ..       Unknown (spotted in newer beacons)
```

IDs 20h-FFh are commonly used; I've received values 2xh..3xh and DDh (from non-nintendo network routers in the neighborhood); no idea if these "Reserved" IDs are somewhere officially documented?

Capability Information

```
Bit0    ESS
Bit1    IBSS
Bit2    CF-Pollable
Bit3    CF-Poll Request
Bit4    Privacy
Bit5    Short Preamble  (IEEE802.11b only)
Bit6    PBCC            (IEEE802.11b only)
Bit7    Channel Agility (IEEE802.11b only)
Bit5-7  Reserved (0) (original IEEE802.11 specs)
Bit8-15 Reserved (0)
```

Listen Interval

```
... used to indicate to the AP how often an STA wakes to listen to Beacon
management frames. The value of this parameter is the STA's Listen Interval
parameter of the MLME-Associate. request primitive and is expressed in
units of Beacon Interval.
```

Reason codes

```
00h Reserved
01h Unspecified reason
02h Previous authentication no longer valid
03h Deauthenticated because sending station is leaving (or has left) IBSS
     or ESS
04h Disassociated due to inactivity
05h Disassociated because AP is unable to handle all currently associated
     stations
06h Class 2 frame received from nonauthenticated station
07h Class 3 frame received from nonassociated station
08h Disassociated because sending station is leaving (or has left) BSS
09h Station requesting (re)association is not authenticated with responding
     station
0Ah..FFFFh Reserved
```

Status codes

```
00h Successful
01h Unspecified failure
02h..09h Reserved
0Ah Cannot support all requested cap's in the Capability Information field
0Bh Reassociation denied due to inability to confirm that association exists
0Ch Association denied due to reason outside the scope of this standard
0Dh Responding station doesn't support the specified authentication algorithm
0Eh Received an Authentication frame with authentication transaction sequence
     number out of expected sequence
```

```
0Fh Authentication rejected because of challenge failure
10h Authentication rejected due to timeout waiting for next frame in sequence
11h Association denied because AP is unable to handle additional associated
     stations
12h Association denied due to requesting station not supporting all of the
     data rates in the BSSBasicRateSet parameter
13h Association denied due to requesting station not supporting
     the Short Preamble option (IEEE802.11b only)
14h Association denied due to requesting station not supporting
     the PBCC Modulation option (IEEE802.11b only)
15h Association denied due to requesting station not supporting
     the Channel Agility option (IEEE802.11b only)
13h-15h Reserved (original IEEE802.11 specs)
16h..FFFFh Reserved
```

DS Wifi IEEE802.11 Control and Data Frames (Type=1 and 2)

Control Frames (Type=1)

All Control Frames have 10-byte or 16-byte headers, depending on the Subtype:

```
Subtype                        Frame Header
0-9 Reserved                   -   -         -      -
A   Power Save (PS)-Poll       FC  AID       BSSID  TA
B   Request To Send (RTS)      FC  Duration  RA     TA
C   Clear To Send (CTS)        FC  Duration  RA     -
D   Acknowledgment (ACK)       FC  Duration  RA     -
E   Contention-Free (CF)-End   FC  Duration  RA     BSSID
F   CF-End + CF-Ack            FC  Duration  RA     BSSID
```

Control Frames do not have a Frame Body, so the Header is directly followed by the FCS.

Data Frames (Type=2)

All Data Frames consist of the following components:

```
FC, Duration/ID, Address 1, Address 2, Address 3, Sequence Control,
Address 4 (only on From DS to DS), Frame Body, FCS.
```

The meaning of the 3 or 4 addresses depends on Frame Control FromDS/ToDS bits:

```
Frame Control    Address 1  Address 2  Address 3  Address 4
From STA to STA  DA         SA         BSSID      -
From DS  to STA  DA         BSSID      SA         -
From STA to DS   BSSID      SA         DA         -
From DS  to DS   RA         TA         DA         SA
```

Frame Control Subtypes for Data Frames (Type=2) are:

```
0   Data
1   Data + CF-Ack
2   Data + CF-Poll
3   Data + CF-Ack + CF-Poll
4   Null function (no data)
5   CF-Ack (no data)
6   CF-Poll (no data)
7   CF-Ack + CF-Poll (no data)
8-F Reserved
```

Note: The Frame Header is 24 bytes (or 30 bytes if with Address 4). The Data in Frame Body is usually starting with LLC stuff, ie. AAh,AAh,03h,00h...

DS Wifi WPA/WPA2 Handshake Messages (EAPOL)

4-Way Handshake and Group Key Handshake messages

For WPA, the Group Key Handshake occurs immediately after the 4-way Handshake (effecitively making it a 6-way handshake). For WPA2, the Group Key is contained in the 4-way Handshake, so the Group Key Handshake

isn't needed.
For both WPA and WPA2, the access point may change the Group Key every once and then, and invoke a new Group Key Handshake for the new key.

EAPOL Key Frame

This is preceeded by a LLC/SNAP header (AAh,AAh,03h,00h,00h,00h,88h,8Eh), the 888Eh in last two bytes indicates that following bytes to contain an EAPOL Key Frame:

```
00h  2    Version/Type (or Type/Version?) (01 03)
02h  2    Length of [04h..end] (005Fh+LEN)                    ;BIG-ENDIAN
04h  1    Descriptor Type (FEh=WPA, 02h=WPA2)
05h  2    Key Information (flags, see below)                  ;BIG-ENDIAN
07h  2    Key Length (0=None, 20h=TKIP, 10h=CCMP, 05h/0Dh=WEP)  ;BIG-ENDIAN
09h  8    Key Replay Counter (usually 0 or 1 in first message)  ;BIG-ENDIAN
11h  32   Key Nonce (ANonce/SNonce)
31h  16   Key Data IV (RC4 uses IV+KEK) (not used for AES-Key-Wrap)
41h  8    Key RSC (TSC/PN) (whatever, for GTK)              ;LITTLE-ENDIAN
49h  8    Reserved (zerofilled)
51h  16   Key MIC on [00h..end] (with MIC initially zerofilled)       ;HMAC
61h  2    Key Data Length (LEN) (00 nn)                       ;BIG-ENDIAN
63h  LEN  Key Data (can be encrypted in certain messages)
```

Key Information flags

```
0-2   Key Descriptor Version (1=WPA/MD5/RC4, 2=WPA2/SHA1/AESkeywrap)
3     Key Type (0=Group, 1=Pairwise)
4-5   Reserved (0) or WPA Group Key Index (1 or 2) (zero for WPA2)
6     Install  (0=No, 1=Yes, configure temporal key)
7     Key Ack  (0=No, 1=Yes, AP wants a reply; with same Key Replay Counter)
8     Key MIC  (0=No, 1=Yes, key frame contains MIC)
9     Secure   (0=No, 1=Yes, initial key-exchange complete)
10    Error    (0=No, 1=Yes, MIC failure and Request=1)
11    Request  (0=No, 1=Yes, request AP to invoke a new handshake)
12    Encrypted(0=No, 1=Yes, Key Data is encrypted; via RC4 or AESkeywrap)
13-15 Reserved (0)
```

Key Data

Key Data does usually consist of element(s) in following form:

```
00h 1   Element ID (for WPA: DDh=RSNIE - for WPA2: 30h=RSNIE, DDh=KDE)
01h 1   Element Length of [02h..end]
02h ..  Element Data (OUI's etc.)
```

WPA and WPA2 assign different meanings to Element ID value DDh. The 3-byte maker prefix in OUI values does also differ for WPA/WPA2. And, the meaning of the OUI's varies depending on WHERE they are used (eg. 00-0F-AC-02 could be a Cipher Suite entry, or an Authentication entry):

```
EAPOL Descriptor Type values
WPA          WPA2         Meaning
FEh          02h          Indicates if ElementIDs and OUIs are WPA or WPA2
EAPOL Key Information flags/values
0089h        008Ah        Handshake #1  ;\
0109h        010Ah        Handshake #2  ; 4-way Handshake
01C9h        13CAh        Handshake #3  ;
0109h(again) 030Ah        Handshake #4  ;/
0391h/03A1h  1382h        Handshake #5  ;\Group Key Handshake
0311h/0321h  0302h        Handshake #6  ;/
EAPOL Key Data Element IDs
DDh          30h          Element ID for RSNIE (Robust Network Security info)
-            DDh          Element ID for KDE (Key Data Encapsulation)
-            DDh          Element ID for padding (followed by 00h-bytes)
RSNIE Prefix OUI's (WPA only):
00-50-F2-01  -            Element Vendor OUI for RSNIE
RSNIE Group Cipher suite selector OUI's (aka Multicast):
00-50-F2-01  00-0F-AC-01  RSNIE Group Cipher WEP-40  (default for US/NSA)
00-50-F2-02  00-0F-AC-02  RSNIE Group Cipher TKIP    (default for WPA)
00-50-F2-04  00-0F-AC-04  RSNIE Group Cipher CCMP    (default for WPA2)
```

```
00-50-F2-05  00-0F-AC-05  RSNIE Group Cipher WEP-104 (default for WEP)
RSNIE Pairwise Cipher suite selector OUI's (aka Unicast):
00-50-F2-00  00-0F-AC-00  RSNIE Pairwise Cipher None (WEP, Group Cipher only)
00-50-F2-02  00-0F-AC-02  RSNIE Pairwise Cipher TKIP (default for WPA)
00-50-F2-04  00-0F-AC-04  RSNIE Pairwise Cipher CCMP (default for WPA2)
RSNIE Authentication AKM suite selector OUI's :
00-50-F2-01  00-0F-AC-01  RSNIE Authentication over IEEE 802.1X (radius?)
00-50-F2-02  00-0F-AC-02  RSNIE Authentication over PSK (default/home use)
KDE Key Data Encapsulation OUI's (WPA2 only):
-            00-0F-AC-01  KDE GTK         (followed by 2+N bytes)
-            00-0F-AC-02  KDE STAKey      (followed by 2+6+N bytes)
-            00-0F-AC-03  KDE MAC address (followed by 6 bytes)
-            00-0F-AC-04  KDE PMKID       (followed by 16 bytes)
```

RSNIE (Robust Network Security Information Element)

RSNIE allows to exchange information about the desired/supported encryption methods. RSNIE is found in handshake message 2/3 (and also in Beacons). In most cases one could just ignore the RSNIE stuff (eg. if Wifi FLASH is already configured to use a specific cipher type). A special case might be networks that use a weaker Group Cipher (for older devices) combined with offering stronger Pairwise Ciphers (for devices that support them).

```
WPA2 RSNIE (Robust Network Security Information Element):
00h  1  Element ID (30h=RSNIE for WPA2)
01h  1  Element Len of [02h..end] (usually 14h)
02h  2  RSNIE Version 1                       (01 00)     ;WHATEVER-ENDIAN?
04h  4  RSNIE Group Cipher Suite OUI (CCMP)   (00 0F AC 04)
08h  2  RSNIE Pairwise Cipher Suite Count (1) (01 00)      ;LITTLE-ENDIAN
0Ah  4  RSNIE Pairwise Cipher Suite OUI (CCMP) (00 0F AC 04)
0Eh  2  RSNIE Authentication Count (1)        (01 00)      ;LITTLE-ENDIAN
10h  4  RSNIE Authentication OUI (PSK)        (00 0F AC 02)
14h  2  RSNIE Capabilities                    (00 00)      ;LITTLE-ENDIAN?
16h (2) RSNIE Optional PMKID Count  ;\usually none such     ;LITTLE-ENDIAN
18h (16)RSNIE Optional PMKID's      ;/
WPA RSNIE (Robust Network Security Information Element):
00h  1  Element ID (DDh=Vendor/RSNIE for WPA)
01h  1  Element Len of [02h..end] (usually 16h or 18h)
02h  4  Element Vendor OUI for RSNIE  (00 50 F2 01)       ;<-- WPA only
06h  2  RSNIE Version value? (1)      (01 00)              ;WHATEVER-ENDIAN?
08h  4  RSNIE Mcast OUI (TKIP)        (00 50 F2 02)
0Ch  2  RSNIE Ucast Count (1)         (01 00)              ;LITTLE-ENDIAN
0Eh  4  RSNIE Ucast OUI (TKIP)        (00 50 F2 02)
12h  2  RSNIE Auth AKM Count (1)      (01 00)              ;LITTLE-ENDIAN
14h  4  RSNIE Auth AKM OUI (PSK)      (00 50 F2 02)
18h (2) RSNIE Capabilities maybe?     (00 00)              ;LITTLE-ENDIAN?
RSN Capabilities flags (usually 0000h) (also spotted: 0C 00):
0   RSN Pre-Auth capabilities
1   RSN No Pairwise capabilities
2-3 RSN PTKSA Replay Counters (0..3 = 1,2,4,16 replay counters)
4-5 RSN GTKSA Replay Counters (0..3 = 1,2,4,16 replay counters)
6   Managment Frame Protection Required
7   Managment Frame Protection Capable
8   Joint Multi-band RSNA
9   PeerKey Enabled
10  SPP A-MSDU Capable
11  SPP A-MSDU Required
12  PBAC
13  Ext Key ID for Unicast
14-15 Reserved (0)
```

KDE Key Data Encapsulation (or raw data for WPA)

KDE (or WPA's raw data) is mainly useful for transferring the GTK group key. Note that Key Data in WPA2 message 3 contains both RSNIE and KDE GTK (and KDE padding). Whilst the raw data in WPA cannot be mixed RSNIE.

```
WPA2 KDE GTK (Key Data Encapsulation for Group Key, in encrypted Key Data):
00h  1   Element ID (DDh=KDE for WPA2)
01h  1   Element Len (16h)
02h  4   KDE OUI GTK (00-0F-AC-01) (occurs in message 3/5)
06h  1   KDE GTK Key ID (01h or 02h)      ;bit2: Tx ?
07h  1   KDE GTK Reserved (00h)
08h  16  KDE GTK Key GTK (for Key ID from above byte [06h])
WPA2 KDE PKMID (Key Data Encapsulation for PKMID) (optional, not needed):
00h  1   Element ID (DDh=KDE for WPA2)
01h  1   Element Len (14h)
02h  4   KDE OUI PMKID (00-0F-AC-04) (optionally occurs in message 1)
06h  16  KDE PMKID (useless checksum on PMK, sometimes exposed in message 1)
WPA2 KDE Padding (for padding Key Data to Nx8 bytes for AES-Key-wrap):
00h  1   Element ID (DDh=KDE for WPA2)
01h  0-6 Padding (00h) (aka Element Len=00h)
WPA GTK (raw Group Key; without Element ID or KDE-style encapsulation):
00h  16  Key GTK (for Key ID from Key Information bit4-5) (in message 5)
```

DS Wifi WPA/WPA2 Keys and MICs

Summary of WPA/WPA2 Keys

```
PSK  Preshared Key (based on password and SSID)
PMK  Pairwise Master Key (same as PSK)
PTK  Pairwise Transient Key (based on PMK, AA, SPA, ANonce, SNonce)
KCK  EAPOL Key Confirmation Key (PTK.bit0..127)    ;for handshake MIC's
KEK  EAPOL Key Encryption Key   (PTK.bit128..255)  ;for handshake Key Data
TK   Temporal Key (TKIP:PTK.bit256..511, CCMP:PTK.bit256..383)
GMK  Group Master Key (don't care, used only internally by the access point)
GTK  Group Transient Key (for multicast/broadcast) (based on GMK, AA, GNonce)
```

Summary of WPA/WPA2 Seeds

```
password  ASCII password for the Wifi network
SSID      ASCII name of access point
AA        MAC address of access point (BSSID)
SPA       MAC address of DSi console
Anonce    Random number from access point (handshake message #1 and #3)
Snonce    Random number from console      (handshake message #2)
Gnonce    Random number internally used by access point (don't care)
```

Summary of WPA/WPA2 Checksums

```
MIC       Message Integrity Code, checksum on EAPOL messages
PMKID     PMK ID, checksum on PMK and AA, SPA (optional, don't care)
```

PRF(key,keylen, src,srclen, dst,dstlen, numrounds) ;Pseudo-Random Function

```
for i=0 to (dstlen-1)/14
  call SHA1HMAC(src,srclen, key,keylen, tmpdst)
  tmpsum[0..13] = tmpdst[0..13]
  for j=1 to numrounds-1                      ;only if numrounds>1
    tmpsrc[0..13] = tmpdst[0..13], tmpsrclen=14
    call SHA1HMAC(tmpsrc,tmpsrclen, key,keylen, tmpdst)
    tmpsum[0..13] = tmpsum[0..13] XOR tmpdst[0..13]
  next j
  src[srclen-1] = src[srclen-1] + 01h          ;increase last byte of src
  len=min(14,(dstlen-i*14))
  dst[i*14+(0..(len-1))] = tmpsum[0..(len-1)]
next i
src[srclen-1] = src[srclen-1] - (dstlen+13)/14  ;undo increments, if desired
```

Note: This is using SHA1HMAC for both WPA and WPA2 (unlike the MIC, which is computed via MD5HMAC for WPA, and SHA1HMAC for WPA2).

PSK (Preshared Key) aka PMK (Pairwise Master Key)

```
key = password,                      keylen = len(password)  ;ASCII string
src = ssid + bytes(00h,00h,00h,01h), srclen = len(ssid)+4    ;ASCII string
dst = PSK,                           dstlen = 32, numrounds=4096
call PRF(key,keylen, src,srclen, dst,dstlen, numrounds)
PMK=PSK
```

Computing the PSK/PMK requires 2x4096 SHA1HMAC calculations, which can be quite slow. The speed could be doubled by pre-computing the SHA1 states for "key XOR 36h's" and "key XOR 5Ch's". And, the PSK/PMK needs to be calculated only when changing the password/ssid (the DSi stores the PSK in Wifi-FLASH).

PTK (Pairwise Transient Key) and KCK/KEK/TK

```
src[0..21]  = "Pairwise key expansion"
src[22]     = byte(00h)
src[23..28] = min(AA,SPA)         ;\MAC addresses (AA=BSSID, SPA=console)
src[29..34] = max(AA,SPA)         ;/
src[35..66] = min(ANonce,SNonce)  ;\nonces from 4-way handshake message 1+2
src[67..98] = max(ANonce,SNonce)  ;/
src[99]     = byte(00h)
srclen = 22+1+6+6+32+32+1 = 100
key=PSK, keylen=32, numrounds=1
dst=PTK, dstlen=64  ;WPA needs dstlen=64 (WPA2 would also work with len=48)
call PRF(key,keylen, src,srclen, dst,dstlen, numrounds)
KCK    = PTK[00h..0Fh]   ;-for EAPOL handshake MIC checksums
KEK    = PTK[10h..1Fh]   ;-for EAPOL handshake Key Data decryption
TK.key = PTK[20h..2Fh]   ;-for data packets
TX.tx  = PTK[30h..37h]   ;\needed for WPA/TKIP only (not WPA2/AES)
TX.rx  = PTK[38h..3Fh]   ;/
TK.keyindex = 0
```

GTK (Group Transient Key)

```
GTK.key = GTK[00h..0Fh]  ;-for data packets
GTX.tx  = GTK[10h..17h]  ;\needed for WPA/TKIP only (not WPA2/AES)
GTX.rx  = GTK[18h..1Fh]  ;/
GTK.keyindex = 1 or 2      ;WPA: from EAPOL Key Information bit4-5
GTK.keyindex = 1 or 2      ;WPA2: from EAPOL Key Data KDE entry
```

MIC (Message Integrity Code)

```
oldmic = EAPOL[51h..60h]
EAPOL[51h..60h] = zerofill
src=EAPOL, srclen=EAPOL[02h]*100h+EAPOL[03h]
key=KCK, keylen=16
if (EAPOL[06h] AND 07h)=1 then call MD5HMAC(src,srclen, key,keylen, dst)
if (EAPOL[06h] AND 07h)=2 then call SHA1HMAC(src,srclen, key,keylen, dst)
newmic = dst[0..0Fh]      ;16-byte MD5 result, or first 16byte of SHA1 result
EAPOL[51h..60h] = newmic
if newmic <> oldmic then error  ;when verifying MIC
```

PMKID (whatever ID, optional/useless, don't care)

```
key=PMK, keylen=32
src[0..7]   = "PMK Name"
src[8..13]  = AA   ;aka MAC address of access point (BSSID)
src[14..19] = SPA  ;aka MAC address of console
srclen = 8+6+6 = 20
call SHA1HMAC(src,srclen, key,keylen, dst)
PMKID = dst[0..0Fh]   ;first 16byte of SHA1 result
```

Note: SHA1HMAC is used for WPA2. Unknown if WPA uses SHA1HMAC, or MD5HMAC, or if WPA does have PMKIDs at all.

Deriving GTK (done internally by access point, don't care)

```
src[0..18]  = "Group key expansion"
```

```
src[19]     = byte(00h)
src[20..25] = AA                ;MAC address (AA=BSSID)
src[26..57] = GNonce            ;whaever random/timer/index
src[58]     = byte(00h)
srclen = 19+1+6+32+1 = 59
key=GMK, keylen=32, numrounds=1  ;whatever random key
dst=GTK, dstlen=32
call PRF(key,keylen, src,srclen, dst,dstlen, numrounds)
```

This is the recommended way for creating the GTK, anyways, this is done internally by the access point, and everybody else doesn't need to worry about how the GTK was generated (ie. one just uses the GTK received in EAPOL messages).

DS Wifi WPA/WPA2 Encryption

Summary of Encryption/Decryption functions

```
Encrypt/Decrypt WPA/WEP packets     --> RC4 (Rivest Cipher 4 aka ARC4)
Encrypt/Decrypt WPA EAPOL key data  --> RC4 (Rivest Cipher 4 aka ARC4)
Encrypt/Decrypt WPA2 EAPOL key data --> AES-Key-Wrap/Unwrap
Encrypt/Decrypt WPA2 packets        --> AES-CCMP (AES-CTR-with-CBC-MAC)
```

RC4 (Rivest Cipher 4 aka ARC4) (WEP/WPA)

Both NDS-Wifi and DSi-Wifi have RC4 hardware support for encrypting and decrypting WEP/WPA packets, however, that's working only for full packets (not for the Key Data field in WPA EAPOL packets, so both NDS and DSi will require a RC4 software implementation for the EAPOL part).

```
RC4(src,dst,len,preskip,key,keylen):
for i=0 to FFh, sbox[i]=i, next i             ;-clear sbox
j=0                                           ;\
for i=0 to FFh                                ;
  j=(j+sbox[i]+key[i mod keylen]) and FFh     ; apply key
  swap(sbox[i],sbox[j]                        ;
next i                                        ;/
i=0, j=0
for k=1 to preskip+len
  i=(i+1) and FFh, j=(j+sbox[i]) and FFh, swap(sbox[i],sbox[j])
  if preskip>0 then preskip=preskip-1
  else [dst]=[src] xor sbox[(sbox[i]+sbox[j]) and FFh], dst=dst+1, src=src+1
next k
parameters for WEP/WPA packets (done by hardware):
key=iv(3)+password(5/13), keylen=3+5/13           ;WEP Key=WEP.IV+Password
key=iv(3)+from PTK???,    keylen=3+???            ;WPA Key=WEP.IV+???
src=data(n)+icv(4), srclen=n+4                    ;src, for WEP
src=data(n)+mic(8)+icv(4), srclen=n+8+4           ;src, for WPA
preskip=0
parameters for WPA EAPOL key data (requires software implementation):
key=EAPOL[31h..40h]+KEK[00h..0Fh], keylen=10h+10h ;Key = EAPOL Key IV + KEK
src=EAPOL+63h, srclen=bigendian(EAPOL[61h])       ;src, for WPA
preskip=100h
```

AES (Advanced Encryption Standard) (WPA2, DSi only)

NDS-Wifi doesn't have any AES hardware support at all, and a software implementation doesn't work because NDS-Wifi automatically applies WEP encryption (and adds/removes the WEP.IV field) when seeing the encryption flag in the frame header (and trying to disable the WEP hardware seems to cause encrypted packets not to be received at all).

DSi-Wifi has hardware support for AES-CCMP encrypted packets, however, AES-Key-Unwrap for Key Data in WPA2 EAPOL packets isn't supported by hardware, and does thus require a software implementation (the ARM7 AES hardware doesn't help because it supports AES-CTR and AES-CCM only).

```
AES-Key-Wrap/Unwrap(src,dst,len,key,keylen,mode) (for WPA2 EAPOL Key Data)
if (len and 7)<>0 then error  ;must be multiple of 8            ;-verify len
aes_setkey(mode,key,keylen)                                     ;-init key
```

```
if mode=ENCRYPT and [src+00h..07h]<>A6A6A6A6A6A6A6A6h then error ;-verify IV
if mode=ENCRYPT then org=dst+8, count=1                   ;-for wrap
if mode=DECRYPT then org=dst+len-8, count=((len-8)/8)*6   ;-for unwrap
[dst+0..len-1] = [src+0..len-1]                  ;copy IV+DATA to dst
[tmp+00h..07h] = [dst+00h..07h]                  ;read IV from dst+0
for i=1 to 6
  ptr=org
  for j=1 to (len-8)/8
    [tmp+08h..0Fh] = [ptr+00h..07h]              ;read DATA from dst+index
    if mode=ENCRYPT then aes_crypt_block(ENCRYPT,tmp,tmp)  ;encrypt tmp
    [tmp+07h]=[tmp+07h] xor count                ;adjust byte[7]
    if mode=DECRYPT then aes_crypt_block(DECRYPT,tmp,tmp)  ;decrypt tmp
    [ptr+00h..07h] = [tmp+08h..0Fh]              ;writeback DATA to dst+index
    if mode=ENCRYPT then ptr=ptr+8, count=count+1
    if mode=DECRYPT then ptr=ptr-8, count=count-1
  next j
next i
[dst+00h..07h] = [tmp+00h..07h]                  ;writeback IV to dst+0
if mode=DECRYPT and [dst+00h..07h]<>A6A6A6A6A6A6A6A6h then error ;-verify IV
Parameters for Wrap/Unwrap:
mode=ENCRYPT  ;<-- for Wrap (encrypt, used by access points)
mode=DECRYPT  ;<-- for Unwrap (decrypt, used by clients)
key=KEK, keylen=10h bytes (128bit)
src=EAPOL+63h, srclen=bigendian(EAPOL[61h])
```

The "aes_setkey" and "aes_crypt_block" functions are same as the BIG-ENDIAN implementations described in DSi AES chapter:
DSi Advanced Encryption Standard (AES)

Access Points / Keys

Keys for different access points are stored in Wifi-FLASH. Access Point 1-3 can be configured as Open/WEP only, and Access Point 4-6 can be Open/WEP/WPA/WPA2 (these extra entries exist on DSi only, the standard NDS firmware doesn't have free memory for storing access point 4-6).
DS Firmware Wifi Internet Access Points

WPA/TKIP Encrypted Packets

```
.. MAC Header         ;-Normal Header
1  TSC1               ;\                   WEPSeed[1]=(TSC1 OR 20h) AND 7Fh
1  WEPSeed[1]         ; WEP IV and Flags
1  TSC0 (LSB)         ; (Flags: bit0-4=Rsvd, bit5=ExtIV, bit6-7=KeyID)
1  Flags              ;/    (bit5: 0=No/WEP, 1=Yes/TKIP)
1  TSC2               ;\
1  TSC3               ; WPA Extended IV
1  TSC4               ;
1  TSC5 (MSB)         ;/
.. Data               ;-Normal Data           ;\
8  MIC                ;-WPA MIC "Michael"     ; encrypted area
4  ICV                ;-WEP ICV               ;/
4  FCS                ;-Normal FCS
```

WPA2/AES-CCMP Encrypted Packets

```
.. MAC Header         ;-Normal Header
1  PN0 (LSB)          ;\
1  PN1                ; CCMP Header (IV and Flags)
1  Rsvd               ; (Flags: bit0-4=Rsvd, bit5=ExtIV, bit6-7=KeyID)
1  Flags              ;     (bit5: 0=No/WEP, 1=Yes/TKIP)
1  PN2                ;
1  PN3                ;
1  PN4                ;
1  PN5 (MSB)          ;/
.. Data               ;-Normal Data           ;\encrypted area
8  MIC                ;-CCMP MIC "AES MAC?"   ;/
4  FCS                ;-Normal FCS
```

CCMP related: A2 (MPDU address 2), AAD, PN, Priority field of MPDU.

TKIP MIC

```
6  DA
6  SA
1  Priority (0) (reserved for future)
3  Zero (0) (also reserved for future)
.. Data
8  MIC (M0..M7) (aka L0..L3, R0..R3)
```

TKIP mixing

```
TTAK     = Phase1 (TK, TA, TSC)
WEP seed = Phase2 (TTAK, TK, TSC)
```

DS Wifi FFC ID

Wifi FCC ID

The wifi hardware has been registered by Mitsume, Hon Hai, and Nintendo themselves:

```
https://fccid.io/BKE - Nintendo
https://fccid.io/EW4 - Mitsumi
https://fccid.io/MCL - Hon Hai (Foxconn)
```

The above webpages include some unrelated stuff (like bluetooth), and some duplicate entries. The wifi/console related entries are:

```
https://fccid.io/EW4-AGBWA      GBA        ;\GBA wireless adaptor
https://fccid.io/EW4-OXYWA      GBA-Micro  ;/(not wifi/wlan compatible)
https://fccid.io/BKENTR001      NDS (non-remove-able board)
https://fccid.io/BKEUSG-001     NDS-Lite (old remove-able board, with MM3155)
https://fccid.io/EW4DWMW006     NDS-Lite (new remove-able board, with MM3218)
https://fccid.io/BKERVL036      Wii
https://fccid.io/EW4DWMW004     Wii (mitsumi) (also W016, and maybe W014 ?)
https://fccid.io/MCLJ27H010     Wii (foxconn) (also H003 ?)
https://fccid.io/EW4DWMW015     DSi (old wifi board, mitsumi)
https://fccid.io/EW4DWMW024     DSi (new wifi board, mitsumi)
https://fccid.io/MCLJ27H020     DSi (new wifi board, foxconn)
https://fccid.io/EW4DWMW028     3DS (mitsumi)
https://fccid.io/MCLJ27H023     3DS (foxconn)
https://fccid.io/MCLJ27H02301   2DS (foxconn)
https://fccid.io/BKERED001      New3DS (on mainboard)
```

The pages include test reports, photos (including some prototypes), and most interestingly, the Mitsumi "Label location" pages are including pinout & signal names for most wifi boards.

DSi Wifi board pinout

https://fccid.io/EW4DWMW024/Label/Label-format-and-location-1137926.pdf
https://fccid.io/EW4DWMW015/Label/Label-Location-1031953.pdf

```
1  MX_SD_CLK     2  GND
3  GND           4  VDD_18
5  SDIO_DATA0    6  VDD_18
7  SDIO_DATA3    8  GND
9  SDIO_DATA1    10 VDD_33
11 SDIO_CMD      12 VDD_33
13 SDIO_DATA2    14 GND
15 JTAG_TDO      16 ATH_TX_H
17 JTAG_TMS      18 SYS_RST_L
19 GND           20 JTAG_TDI
21 CLK32k        22 JTAG_TCK
23 GND           24 JTAG_TRST_L
25 NC(VDD28_TP)  26 SEL_ATH_L
27 SPI_CS2       28 W_B
29 BBP_SLEEP     30 SPI_CLK
```

31	RF_SLEEP	32	SPI_DO MISO
33	RF_SCS	34	SPI_DI MOSI
35	BBP_SCS	36	CCA
37	BB_RF_SDO	38	RXPE
39	BB_RF_SDI	40	TRDATA
41	BB_RF_SCLK	42	GND
43	NC(VDD18_TP)	44	TRCLK
45	GND	46	TRRDY
47	MCLK	48	TXPE
49	GND	50	RESET

DS-Lite Wifi board DWM-W006 pinout

https://fccid.io/EW4DWMW006/Label/ID-label-format-and-location-706511.pdf

1	GND	16	+3.3V
2	TXPE	17	GND
3	RXPE	18	RF_SCS
4	CCA	19	BBP_SLEEP
5	TRRDY	20	BBP_SCS
6	GND	21	RF_SLEEP
7	TRCLK	22	RESET
8	TRDATA	23	GND
9	GND	24	SPI_CLK
10	BB_RF_SDO	25	SPI_DI MOSI
11	BB_RF_SDI	26	SPI_DO MISO
12	BB_RF_SCLK	27	W_B
13	GND	28	SPI_CS2
14	MCLK	29	LD ?
15	GND	30	GND

Wii Wifi board

https://fccid.io/EW4DWMW004/Label/ID-Label-670426.pdf

1	GND	12	VDD3.3
2	SDIO_DATA_2	13	GND
3	SDIO_DATA_1	14	GPIO_0
4	GND	15	GND
5	SDIO_CLK	16	SDIO_DATA_3
6	GND	17	SDIO_DATA_0
7	GPIO_1	18	GND
8	GND	19	SDIO_CMD
9	N.C.(VDD1.8)	20	GND
10	N.C.(VDD1.8)	21	ANT_A (MAIN)
11	VDD3.3	22	ANT_B (AUX)

3DS Wifi Board DWM-W028

https://fccid.io/EW4DWMW028/Label/ID-label-location-1272988.pdf

1	MCLK	2	RF_CSRF
3	GND	4	BB_CSBB
5	RXPE	6	BB_RF_SDIN
7	TXPE	8	BB_RF_SDOUT
9	CCA	10	BB_RF_SCK
11	TRDATA	12	GND
13	TRCLK	14	BBP_SLEEP_L
15	TRRDY	16	RF_SLEEP_L
17	TRST_L	18	SEL_ATH_L
19	GND	20	GND
21	SDIO_DATA_0	22	JTAG_TDO
23	SDIO_DATA_1	24	JTAG_TMS
25	SDIO_DATA_2	26	JTAG_TDI
27	SDIO_DATA_3	28	JTAG_TCK
29	GND	30	SPI_CS2
31	SDIO_CLK	32	W_B
33	GND	34	SPI_CLK
35	SDIO_CMD	36	SPI_DO MISO

```
  37 UART_TXD     38 SPI_DI   MOSI
  39 UART_RXD     40 SYS_RST_L
  41 GND          42 ATH_TX_H
  43 CLK32k       44 RESET
  45 GND          46 GND
  47 VDD_18       48 VDD_33
  49 VDD_18       50 VDD_33
```
Functionally same as DSi, but not pin-compatible, and with two UART pins (instead of the NC pins).

GBA-Micro: device has two channels only: 2426.248MHz and 2456.576MHz.

DS Wifi Dslink/Wifiboot Protocol

Below is the transfer protocol for nocash "wifiboot" and devkitpro "dslink".
All UDP/TCP message are using Port 17491 (4453h).

Wifiboot UDP announce/reply and listen/connect/accept
```
  PC sends UPD announce message (repeatedly, as broadcast)  ;\
  console does TCP listen (if it isn't already listening)   ; UDP and
  console sends UDP reply message (repeatedly, to PC)       ; listen/accept
  PC does TCP connect (upon UDP reply)                      ;
  console does TCP accept (upon TCP connect)                ;/
```
The announce/reply strings in UDP messages depend on the file type:
```
  type      NDS/DSi    3DS.firm       3DS.3dsx
  announce  "dsboot"   "3dsfirmboot"  "3dsboot"
  reply     "bootds"   "bootfirm3ds"  "boot3ds"
```
After passing through the announce/reply phase, the actual data is transferred via TCP:

Wifiboot TCP blocks for NDS/DSi
```
  PC sends NDS header (170h bytes)                 ;SMALLER ;\
  PC sends Info Block (90h bytes)                      ;NEW ;
  console sends 32bit response word (4 bytes)                ;
  PC sends Icon/Title (optional, if response.bit17)    ;NEW ;
  PC sends DSi header (1000h bytes, if response.bit16)       ; TCP transfer
  PC sends ARM7 bootcode                                     ;
  PC sends ARM9 bootcode                                     ;
  PC sends ARM7i bootcode (optional, if response.bit16)      ;
  PC sends ARM9i bootcode (optional, if response.bit16)      ;
  PC sends commandline 32bit length (00000000h=none)         ;
  PC sends commandline string (if any)                       ;/
```

Wifiboot TCP blocks for 3DS.firm
```
  PC sends FIRM header (200h bytes)                          ;\
  PC sends Info Block (90h bytes)                            ;
  console sends 32bit response word (4 bytes)                ;
  PC sends Icon/Title (optional, if response.bit17)          ;
  PC sends Logo       (optional, if response.bit18)          ; TCP transfer
  PC sends Banner     (optional, if response.bit19)          ;
  PC sends FIRM section 0                                    ;
  PC sends FIRM section 1                                    ;
  PC sends FIRM section 2                                    ;
  PC sends FIRM section 3                                    ;
  PC sends commandline 32bit length (0=none)                 ;
  PC sends commandline string (if any)                       ;/
```

Wifiboot TCP blocks for 3DS.3dsx
Not supported by wifiboot, but supported by devkitpro. The exact protocol is unknown, it seems to be similar to the above protocols, but with deflate compression (see "3dsboot" source code for details).

Response word (32bit)
This did originally contain 32bit error flags, and was later changed to contain request flags in bit16 and up (nonetheless, uploaders should keep treating any unexpected or unsupported request flags as errors).

```
0-x   Error flags
16    Request DSi header and ARM7i/ARM9i blocks (DSi)
17    Request Icon/Title (NDS/DSi/3DS)
18    Request Logo       (3DS)
19    Request Banner     (3DS)
20-31 Reserved (0)
```

The downloader should set request flags only if it wants to receive extra data, and only if the header or info block indicates that data to be available.

Info Block (90h bytes)
This did formerly contain NDS file bytes 170h..1FFh (ie. as if the cartheader were 200h bytes in size).

```
00h 8   Overall ID "BootINFO" (if other: ignore all entries below)
08h 24  Uploader name/version, zeropadded (eg. "nocash wifiboot v2.6")
20h 1   Time Seconds    (BCD, 00h..59h)       ;\
21h 1   Time Minutes    (BCD, 00h..59h)       ; current time (local timezone)
22h 1   Time Hours      (BCD, 00h..23h)       ; (for updating lost RTC time)
23h 1   Time DayOfWeek  (0..6, 0=Monday)      ; (or all zeroes = none)
24h 1   Time Day        (BCD, 01h..31h)       ;
25h 1   Time Month      (BCD, 01h..12h)       ;
26h 1   Time Year       (BCD, 00h..99h)       ;
27h 1   Time Century    (BCD, 00h..99h)       ;/
28h 4   Icon/Title Size (0=None, 840h/940h/A40h/23C0h=NDS/DSi, 36C0h=3DS)
2Ch 4   Logo Size       (0=None, Other=3DS only)
30h 4   Banner Size     (0=None, Other=3DS only)
34h 5Ch Reserved (0)
```

DS Xboo

The DS Xboo cable allows to upload NDS ROM-Images (max 3.9MBytes) to the console via parallel port connection. Should be the best, simpliest, easiest, and fastest way to test code on real hardware. And, at a relative decent price of 11 cents per diode it should be by far the least expensive way. You'll have to touch classic tools (screwdrivers, knifes, saws, tweezers, and solder) which will probably scare most of you to hell.

DS XBOO Connection Schematic

```
Console Pin/Names             Parallel Port Pin/Names
RFU.9    FMW.1 D    ---|>|--- DSUB.14    CNTR.14    AutoLF
RFU.6    FMW.2 C    ---|>|--- DSUB.1     CNTR.1     Strobe
RFU.10   FMW.3 /RES ---|>|--- DSUB.16    CNTR.31    Init
RFU.7    FMW.4 /S   ---|>|--- DSUB.17    CNTR.36    Select
RFU.5    FMW.5 /W   --. SL1A  -           -          N.C.
RFU.28   FMW.6 VCC  __| SL1B  -           -          N.C.
RFU.2,12 FMW.7 VSS  --------- DSUB.18-25 CNTR.19-30 Ground
RFU.8    FMW.8 Q    --------- DSUB.11    CNTR.11    Busy
P00 Joypad-A        ---|>|--- DSUB.2     CNTR.2     D0
P01 Joypad-B        ---|>|--- DSUB.3     CNTR.3     D1
P02 Joypad-Select   ---|>|--- DSUB.4     CNTR.4     D2
P03 Joypad-Start    ---|>|--- DSUB.5     CNTR.5     D3
P04 Joypad-Right    ---|>|--- DSUB.6     CNTR.6     D4
P05 Joypad-Left     ---|>|--- DSUB.7     CNTR.7     D5
P06 Joypad-Up       ---|>|--- DSUB.8     CNTR.8     D6
P07 Joypad-Down     ---|>|--- DSUB.9     CNTR.9     D7
RTC.1 INT aka SI    --------- DSUB.10    CNTR.10    /Ack
```

Parts List: 15 wires, four (DS) or twelve (DS-Lite) "BAT 85" diodes, 1 parallel port socket.

DS XBOO Connection Notes
The Firmware chip (FMW.Pins) hides underneath of the RFU shielding plate, so it'd be easier to connect the

wires to the RFU.Pins (except DS-Lite: The RFU pins are terribly small (and have different pin-numbers), so either using FMW.Pins, or using mainboard vias (see below GIF) would be easier). The easiest way for the /W-to-VCC connection is to shortcut SL1 by putting some solder onto it.
The P00..P07 and INT signals are labeled on the switch-side of the mainboard, however, there should be more room for the cables when connecting them to via's at the bottom-side (except DS-Lite: P01 is found only at switch-side) image below may help to locate that pins,
  `http://problemkaputt.de/nds-pins.gif (GIF-Image, 7.5KBytes)`
At the parallel port side, DSUB.Pins or CNTR.Pins can be used for 25pin DSUB or 36pin Centronics sockets, the latter one allowing to use a standard printer cable.
The ring printed on the diodes is pointing towards parallel port side, the 4 diodes are required to prevent the parallel port to pull-up LOW levels on the NDS side, be sure to use BAT85 diodes, cheaper ones like 1N4148 are loosing too much voltage and won't gain stable LOW levels.
The power managment chip in the DS-Lite simply refuses to react to the Power-On button when P00..P07 are dragged high by the parallel port (even if it is in HighZ state), the 8 diodes in the data-lines are solving that problem (they are required on DS-Lite only, not on original DS).

DS XBOO Operation Notes
The main Upload function is found in no$gba Utility menu, together with further functions in Remote Access sub-menu.
Before uploading anything: download the original firmware, the file is saved as FIRMnnnn.BIN, whereas "nnnn" is equal to the last 16bit of the consoles 48bit MAC address, so Firmware-images from different consoles are having unique filenames. If you don't already have, also download the NDS BIOS, the BIOS contains encryption seed data required to encrypt/decrypt secure area; without having downloaded the BIOS, no$gba will be working only with unencrypted ROM-images. Next, select Patch Firmware to install the nocash firmware.

DS XBOO Troubleshooting
Be sure that the console is switched on, and that the XBOO cable is connected, and that you have selected the correct parallel port in no$gba setup (the "multiboot" options in Various Options screen), and, of course, try avoid to be fiddling with the joypad during uploads.
I've tested the cable on two computers, the overall upload/download stuff should work stable. The firmware access functions - which are required only for (un-)installation - worked only with one of the two computers; try using a different computer/parallel port in case of problems.

Nocash Firmware
The primary purpose is to receive uploaded NDS-images via parallel port connection, additionally it's containing bootmenu and setup screens similar to the original firmware. The user interface is having less cryptic symbols and should be alltogether faster and easier to use. Important Information about Whatever is supported (but it can be disabled). The setup contains a couple of additional options like automatic daylight saving time adjustment.
The bootmenu allows to boot normal NDS and GBA carts, it does additionally allow to boot NDS-images (or older PassMe-images) from flashcards in GBA slot. Furthermore, benefits of asm coding, the nocash firmware occupies less than 32KBytes, allowing to store (and boot) smaller NDS-images in the unused portion of the firmware memory (about 224KBytes), the zero-filled region between cart header and secure area, at 200h..3FFFh, is automatically excluded, so the image may be slightly bigger than the available free memory space.

Missing
Unlike the original firmware, the current version cannot yet boot via WLAN.

DSi Reference

Basic Hardware Features (mostly same as NDS)
NDS Reference
DSi Basic Differences to NDS

New Hardware Features
DSi I/O Map
DSi Control Registers (SCFG)
DSi XpertTeak (DSP)
DSi New Shared WRAM (for ARM7, ARM9, DSP)
DSi New DMA (NDMA)
DSi Microphone and SoundExt
DSi Advanced Encryption Standard (AES)
DSi Cartridge Header
DSi Touchscreen/Sound Controller
DSi I2C Bus
DSi Cameras
DSi SD/MMC Protocol and I/O Ports
DSi SD/MMC Filesystem
DSi Atheros Wifi SDIO Interface
DSi Atheros Wifi Internal Hardware
DSi GPIO Registers
DSi Console IDs
DSi Unknown Registers
DSi Notes
DSi Exploits
DSi Regions

General Info
ARM CPU Reference
BIOS Functions
External Connectors

Credits: http://www.dsibrew.org/wiki/Special:AllPages (now spammed)

DSi Basic Differences to NDS

Interrupts
There are several new interrupt sources in IE/IF registers, plus further ones in new IE2/IF2 registers.
DS Interrupts

Video
Essentially same as for NDS. Some details can be changed in SCFG_EXT. For the 2D Engine, DISPSTAT.Bit6 contains a new "LCD Initialization Ready" flag on both ARM7 and ARM9 side (the bit is checked by DSi System Menu) (the bit is supposedly used at power-up, maybe also for wake-up from certain sleep modes).

BIOS SWI Functions
Some SWI Functions are changed (bugged in some cases), new SHA1 and RSA functions are added, and the initial RAM contents are moved from 27FFxxxh to 2FFFxxxh (with some extra fields, eg. a copy of extended DSi cart header).
BIOS Functions

Revised Hardware Functions
Some hardware features have been slightly revised (for example, the division by 0 flag was fixed). The revised functions can be enabled/disabled via SCFG registers.
DSi Control Registers (SCFG)

NDS Slot / Cartridges
DSi carts are using an extended cart header (1000h bytes), with RSA signature (making it problematic to run unlicensed/homebrew code), the icon/title format has been also extended, and the cartridge protocol contains a new command (command 3Dh, for unlocking extra DSi regions on the cartridge, and for reading new DSi secure area blocks).
The NDS Slot's Reset signal can be controlled by software (required because otherwise one could use only command 3Ch or 3Dh, but not both). The Power supply pin can be also controlled by software (yet not 100% confirmed how?). Moreover, there's new cartridge inserted sensor. And, DSi prototypes did have two NDS slots; DSi retail consoles do have only one slot, but they do still contain prototype relicts internally (like extra registers and extra irq sources for second slot) (there appear to be also unused extra pins on the CPU, but they couldn't be used without desoldering the whole chip).

Enable Bits
One new DSi invention is that setting Enable Bits (eg. for NDMA or CAM registers) is write-protecting the corresponding registers (ie. those registers can be initialized only while the Enable Bits are off).

SPI Bus
The SPI bus works same as on NDS, except, one small change is added support for 8MHz SPI clock (in SPICNT.bit2).

SPI Touchscreen Controller
This chip is working entirely different in DSi mode. It's still accessed via SPI bus, but with some new PAGE/INDEX values.
DSi Touchscreen/Sound Controller
The touchscreen hardware can be switched to NDS compatibility mode.

SPI Power Managment Device
The Power Managment Device contains some changed register, and some new extra registers. Internally, it is actually split to two devices: The power managment chipselect signal connects to U3 and U4 chips. Ie. some SPI registers are processed by U3 (power down, and backlight enable), and others by R4 (audio out and microphone).
Further functions like LED control and backlight brightness are moved to the BPTWL chip, accessed via I2C bus instead of SPI bus - the power LED blink feature (which was used on Wifi access) seems to be no longer working, however, the Wifi LED seems to blink automatically on Wifi access; the changed backlight brightness mechanism shouldn't cause compatibility issuses since that feature is somewhat reserved for being controlled by the firmware.

SPI FLASH Memory (Wifi Calibration, User Settings, Firmware)
This memory does still exist, but it's only 128Kbytes in DSi (instead 256K), and most of it is empty (since the DSi Firmware is stored in the eMMC chip). Later DSi models do even the SPI FLASH capacity reduced to 4Kbytes.

RTC
Should be compatible with NDS. But seems to contain extra registers?
One of the RTC outputs does also seem to supply some (32kHz?) clock to some other mainboard components? [XXX see Seiko S-35199A01 datasheet].

Wifi
Supports new WPA and WPA2 encryption and supports higher transfer rates (via a new SDIO wifi unit). For the old NDS-wifi mode, there are some new control registers:

```
4004020h - SCFG_WL
4004C04h - GPIO_WIFI
BPTWL[30h] - Wifi LED related (also needed to enable Atheros Wifi SDIO)
```

SPI FLASH contains three new access point settings (for WPA/WPA2/proxy support):
DS Firmware Wifi Internet Access Points
The access point configuration can be done via Firmware (unlike as on NDS, where it needed to be done by the games).

GBA Slot
The GBA Slot has been removed. The memory regions and IRQ bits do still exist internally, but the DSi does basically behave as if there is no GBA cartridge inserted. Reading GBA ROM areas does return FFFFh halfwords instead of the usual open bus values though.

NDS Mode
In NDS mode, the DSi is basically working same as NDS: The new extra hardware is disabled, original NDS BIOS ROMs are mapped, and the hardware is simulating the old touchscreen controller.
Nonetheless, there are still a some small differences to real NDS consoles:
- Unlicensed NDS carts don't work (requires RSA, or whitelist for older games)
- GBA Slot is removed (more or less behaves as if no cart inserted)
- DSi ports 4004700h and 4004C0xh can be read (and written?) even in DS mode
- SPI Power Managment has some added/removed/changed registers
- SPI Touchscreen controller doesn't support pressure & temperature
- SPI FLASH exists, but it's smaller, and has extra access point info, etc.
- ARM7 BIOS has only first 20h bytes locked (instead first 1204h bytes)
- Power Button issues a Reset (goes to boot menu) (instead of plain power off)
- Power LED blink feature (for wifi) is replaced by a separate wifi LED
- RTC extra registers (if they do really exist) should exist in DS mode (?)
Unknown: does hot-swapping auto-power-off the nds-cart-slot in nds mode?

DSi I/O Map

DSi Memory Map
The overall memory map is same as on NDS. New/changed areas are:

```
0000000h  64Kbyte ARM7 BIOS   (unlike NDS which had only 16KB)
2000000h  16MByte Main RAM    (unlike NDS which had only 4MB)
3000000h  800Kbyte Shared RAM (unlike NDS which had only 32KB)
4004000h  New DSi I/O Ports
8000000h  Fake GBA Slot (32MB+64KB) (FFh-filled; when mapped to current CPU)
C000000h  Mirror of 16Mbyte Main RAM
D000000h  Open Bus? in retail version, Extra 16Mbyte MainRAM in debug version
FFFF000h  64Kbyte ARM9 BIOS   (unlike NDS which had only 4KB)
```

DSi I/O Maps
The overall DSi I/O Maps are same as on NDS,
DS I/O Maps
additional new/changed registers are:

ARM9 NDS Register that are changed in DSi mode

```
4000004h 2   DISPSTAT (new Bit6, LCD Initialization Ready Flag)
4000204h 2   EXMEMCNT (removed Bit0-7, ie. the GBA-slot related bits)
4000210h 4   IE       (new interrupt sources, removed GBA-slot IRQ)
4000214h 4   IF       (new interrupt sources, removed GBA-slot IRQ)
40021A0h 4   Unknown, nonzero, probably same/silimar as on DSi7 side
40021A4h 4   Unknown, zero, probably same/silimar as on DSi7 side
40021A8h ..
40021Bxh ..
4102010h 4
```

ARM9 DSi System Control

```
  4004000h 2   SCFG_A9ROM DSi - NDS9 - ROM Status (R) [0000h]
  4004004h 2   SCFG_CLK   DSi - NDS9 - New Block Clock Control (R/W)
  4004006h 2   SCFG_RST   DSi - NDS9 - New Block Reset (R/W)
  4004008h 4   SCFG_EXT   DSi - NDS9 - Extended Features (R/W)
  4004010h 2   SCFG_MC    Memory Card Interface Status (16bit) (undocumented)
```

ARM9 DSi New Shared WRAM Bank Control

```
  4004040h 4   MBK1       WRAM-A Slots for Bank 0,1,2,3  ;\Global ARM7+ARM9
  4004044h 4   MBK2       WRAM-B Slots for Bank 0,1,2,3  ; Slot Mapping
  4004048h 4   MBK3       WRAM-B Slots for Bank 4,5,6,7  ; (R or R/W, depending
  400404Ch 4   MBK4       WRAM-C Slots for Bank 0,1,2,3  ; on MBK9 setting)
  4004050h 4   MBK5       WRAM-C Slots for Bank 4,5,6,7  ;/
  4004054h 4   MBK6       WRAM-A Address Range           ;\Local ARM9 Side
  4004058h 4   MBK7       WRAM-B Address Range           ; (R/W)
  400405Ch 4   MBK8       WRAM-C Address Range           ;/
  4004060h 4   MBK9       WRAM-A/B/C Slot Write Protect (R)
```

ARM9 DSi New DMA (NDMA)

```
  4004100h 4   NDMAGCNT   NewDMA Global Control                    ;-Control
  4004104h 4   NDMA0SAD   NewDMA0 Source Address                   ;\
  4004108h 4   NDMA0DAD   NewDMA0 Destination Address              ;
  400410Ch 4   NDMA0TCNT  NewDMA0 Total Length for Repeats         ; NewDMA0
  4004110h 4   NDMA0WCNT  NewDMA0 Logical Block Size               ;
  4004114h 4   NDMA0BCNT  NewDMA0 Block Transfer Timing/Interval   ;
  4004118h 4   NDMA0FDATA NewDMA0 Fill Data                        ;
  400411Ch 4   NDMA0CNT   NewDMA0 Control                          ;/
  4004120h 4   NDMA1SAD                                            ;\
  4004124h 4   NDMA1DAD                                            ;
  4004128h 4   NDMA1TCNT                                           ; NewDMA1
  400412Ch 4   NDMA1WCNT                                           ;
  4004130h 4   NDMA1BCNT                                           ;
  4004134h 4   NDMA1FDATA                                          ;
  4004138h 4   NDMA1CNT                                            ;/
  400413Ch 4   NDMA2SAD                                            ;\
  4004140h 4   NDMA2DAD                                            ;
  4004144h 4   NDMA2TCNT                                           ; NewDMA2
  4004148h 4   NDMA2WCNT                                           ;
  400414Ch 4   NDMA2BCNT                                           ;
  4004150h 4   NDMA2FDATA                                          ;
  4004154h 4   NDMA2CNT                                            ;/
  4004158h 4   NDMA3SAD                                            ;\
  400415Ch 4   NDMA3DAD                                            ;
  4004160h 4   NDMA3TCNT                                           ; NewDMA3
  4004164h 4   NDMA3WCNT                                           ;
  4004168h 4   NDMA3BCNT                                           ;
  400416Ch 4   NDMA3FDATA                                          ;
  4004170h 4   NDMA3CNT                                            ;/
```

ARM9 DSi Camera Module

```
  4004200h 2   CAM_MCNT   Camera Module Control (16bit)
  4004202h 2   CAM_CNT    Camera Control (16bit)
  4004204h 4   CAM_DAT    Camera Data (32bit)
  4004210h 4   CAM_SOFS   Camera Trimming Starting Position Setting (32bit)
  4004214h 4   CAM_EOFS   Camera Trimming Ending Position Setting (32bit)
```

ARM9 DSi DSP - XpertTeak processor

```
  4004300h 2   DSP_PDATA  DSP Transfer Data
  4004304h 2   DSP_PADR   DSP Transfer Address
  4004308h 2   DSP_PCFG   DSP Configuration
  400430Ch 2   DSP_PSTS   DSP Status
  4004310h 2   DSP_PSEM   DSP ARM9-to-DSP Semaphore          (R/W)
  4004314h 2   DSP_PMASK  DSP DSP-to-ARM9 Semaphore Mask     (R/W)
  4004318h 2   DSP_PCLEAR DSP DSP-to-ARM9 Semaphore Clear    (W)
  400431Ch 2   DSP_SEM    DSP DSP-to-ARM9 Semaphore Data     (R)
  4004320h 2   DSP_CMD0   DSP ARM9-to-DSP Command Register 0 (R/W)
  4004324h 2   DSP_REP0   DSP DSP-to-ARM9 Reply Register 0   (R)
  4004328h 2   DSP_CMD1   DSP ARM9-to-DSP Command Register 1 (R/W)
  400432Ch 2   DSP_REP1   DSP DSP-to-ARM9 Reply Register 1   (R)
```

```
  4004330h 2   DSP_CMD2   DSP ARM9-to-DSP Command Register 2 (R/W)
  4004334h 2   DSP_REP2   DSP DSP-to-ARM9 Reply Register 2   (R)
  4004340h C0h DSP_mirror Mirrors of above 40h-byte DSP register area
```

ARM7 DSi

```
  4000004h 2   DISPSTAT (new Bit6, LCD Initialization Ready Flag) (as DSi9?)
  40001C0h 2   SPICNT   (new Bit2, for 8MHz transfer clock)
  4000204h 2   EXMEMCNT (removed Bit0-7: GBA-slot related bits)   (as DSi9?)
  4000210h 4   IE       (new interrupt sources, removed GBA-slot IRQ)
  4000214h 4   IF       (new interrupt sources, removed GBA-slot IRQ)
  4000218h     IE2      (new register with more new interrupt sources)
  400021Ch     IF2      (new register with more new interrupt sources)
```

ARM7 DSi Maybe 2nd ROM slot (DSi prototypes did have 2 cartridge slots)

```
  40021A0h 4   Unknown, nonzero, probably related to below 40021A4h
  40021A4h 4   Unknown, related to 40001A4h (Gamecard Bus ROMCTRL)
  40021A8h ..
  40021Bxh ..
  4102010h 4
```

ARM7 DSi System Control

```
  4004000h 1   SCFG_A9ROM used by BIOS and SystemFlaw   (bit0,1)
  4004001h 1   SCFG_A7ROM used by BIOS and SystemFlaw   (bit0,1,2)
  4004004h 2   SCFG_CLK7  used by SystemFlaw
  4004006h 2   SCFG_JTAG  Debugger Control
  4004008h 4   SCFG_EXT7  used by SystemFlaw
  4004010h 2   SCFG_MC    Memory Card Interface Control (R/W)  ;\
  4004012h 2   SCFG_CARD_INSERT_DELAY (usually 1988h = 100ms)  ; Game Cartridge
  4004014h 2   SCFG_CARD_PWROFF_DELAY (usually 264Ch = 150ms)  ;/
  4004020h 2   SCFG_WL    Wireless Disable    ;bit0 = wifi?
  4004024h 2   SCFG_OP    Debugger Type (R)   ;bit0-1 = (0=retail, ?=debug)
```

ARM7 DSi New Shared WRAM Bank Control

```
  4004040h 4   MBK1       WRAM-A Slots for Bank 0,1,2,3  ;\
  4004044h 4   MBK2       WRAM-B Slots for Bank 0,1,2,3  ; Global ARM7+ARM9
  4004048h 4   MBK3       WRAM-B Slots for Bank 4,5,6,7  ; Slot Mapping (R)
  400404Ch 4   MBK4       WRAM-C Slots for Bank 0,1,2,3  ; (set on ARM9 side)
  4004050h 4   MBK5       WRAM-C Slots for Bank 4,5,6,7  ;/
  4004054h 4   MBK6       WRAM-A Address Range           ;\Local ARM7 Side
  4004058h 4   MBK7       WRAM-B Address Range           ; (R/W)
  400405Ch 4   MBK8       WRAM-C Address Range           ;/
  4004060h 4   MBK9       WRAM-A/B/C Slot Write Protect (R/W)
```

ARM7 DSi New DMA (NDMA)

```
  4004100h 74h NewDMA (new DMA, as on ARM9i, see there)
```

ARM7 DSi AES Encryption Unit

```
  4004400h 4   AES_CNT    (R/W)
  4004404h 4   AES_BLKCNT (W)
  4004408h 4   AES_WRFIFO (W)
  400440Ch 4   AES_RDFIFO (R)
  4004420h 16  AES_IV     (W)
  4004430h 16  AES_MAC    (W)
  4004440h 48  AES_KEY0   (W) ;used for modcrypt
  4004470h 48  AES_KEY1   (W) ;used for ?
  40044A0h 48  AES_KEY2   (W) ;used for JPEG signatures
  40044D0h 48  AES_KEY3   (W) ;used for eMMC sectors
```

ARM7 DSi I2C Bus

```
  4004500h 1   I2C_DATA
  4004501h 1   I2C_CNT
```

ARM7 DSi Microphone

```
  4004600h 2   MIC_CNT    Microphone Control
  4004604h 4   MIC_DATA   Microphone FIFO
```

ARM7 DSi SNDEX

```
  4004700h 2   SNDEXCNT             <-- can be read even in DS mode!
```

ARM7 DSi SD/MMC Registers for Memory Card access (SD Card and onboard eMMC)

```
  4004800h 2   SD_CMD              Command and Response/Data Type
  4004802h 2   SD_CARD_PORT_SELECT   (SD/MMC:020Fh, SDIO:010Fh)
```

```
  4004804h 4   SD_CMD_PARAM0-1     Argument (32bit, 2 halfwords)
  4004808h 2   SD_STOP_INTERNAL_ACTION
  400480Ah 2   SD_DATA16_BLK_COUNT        "Transfer Block Count"
  400480Ch 16  SD_RESPONSE0-7 (128bit, 8 halfwords)
  400481Ch 4   SD_IRQ_STATUS0-1  ;IRQ Status  (0=ack, 1=req)
  4004820h 4   SD_IRQ_MASK0-1    ;IRQ Disable (0=enable, 1=disable)
  4004824h 2   SD_CARD_CLK_CTL Card Clock Control
  4004826h 2   SD_DATA16_BLK_LEN  Memory Card Transfer Data Length
  4004828h 2   SD_CARD_OPTION Memory Card Option Setup (can be C0FFh)
  400482Ah 2   Fixed always zero?
  400482Ch 4   SD_ERROR_DETAIL_STATUS0-1   Error Detail Status
  4004830h 2   SD_DATA16_FIFO        Data Port  (SD_FIFO?)
  4004832h 2   Fixed always zero?       ;(TC6371AF:BUF1 Data MSBs?)
  4004834h 2   SD_CARD_IRQ_CTL          ;(SD_TRANSACTION_CTL)
  4004836h 2   SD_CARD_IRQ_STAT         ;(SD_CARD_INTERRUPT_CONTROL)
  4004838h 2   SD_CARD_IRQ_MASK         ;(SDCTL_CLK_AND_WAIT_CTL)
  400483Ah 2   Fixed always zero?       ;(SDCTL_SDIO_HOST_INFORMATION)
  400483Ch 2   Fixed always zero?       ;(SDCTL_ERROR_CONTROL)
  400483Eh 2   Fixed always zero?       ;(TC6387XB: LED_CONTROL)
  4004840h 2   Fixed always 003Fh?
  4004842h 2   Fixed always 002Ah?
  4004844h 6Eh Fixed always zerofilled?
  40048B2h 2   Fixed always FFFFh?
  40048B4h 6   Fixed always zerofilled?
  40048BAh 2   Fixed always 0200h?
  40048BCh 1Ch Fixed always zerofilled?
  40048D8h 2   SD_DATA_CTL
  40048DAh 6   Fixed always zerofilled?
  40048E0h 2   SD_SOFT_RESET  Software Reset (bit0=SRST=0=reset)
  40048E2h 2   Fixed always 0009h?      ;(RESERVED2/9, TC6371AF:CORE_REV)
  40048E4h 2   Fixed always zero?
  40048E6h 2   Fixed always zero?       ;(RESERVED3, TC6371AF:BUF_ADR)
  40048E8h 2   Fixed always zero?       ;(TC6371AF:Resp_Header)
  40048EAh 6   Fixed always zerofilled?
  40048F0h 2   Fixed always zero?       ;(RESERVED10)
  40048F2h 2   ? Can be 0003h
  40048F4h 2   ? Can be 0770h
  40048F6h 2   SD_WRPROTECT_2 (R)  ;Write protect for eMMC   (RESERVED4)
  40048F8h 4   SD_EXT_IRQ_STAT0-1  ;Insert/eject for eMMC    (RESERVED5-6)
  40048FCh 4   SD_EXT_IRQ_MASK0-1       ;(TC6371AF:Revision) (RESERVED7-8)
  4004900h 2   SD_DATA32_IRQ
  4004902h 2   Fixed always zero?
  4004904h 2   SD_DATA32_BLK_LEN
  4004906h 2   Fixed always zero?
  4004908h 2   SD_DATA32_BLK_COUNT
  400490Ah 2   Fixed always zero?
  400490Ch 4   SD_DATA32_FIFO
  4004910h F0h Fixed always zerofilled?
```

ARM7 DSi SD/MMC Registers for SDIO access (for Atheros Wifi)

```
  4004A00h 512 SDIO_xxx (same as SD_xxx at 4004800h..40049FFh, see there)
  4004A02h 2   SDIO_CARD_PORT_SELECT (slightly different than 4004802h)
```

ARM7 DSi General Purpose I/O (GPIO) (headphone connect, power button)

```
  4004C00h 1   GPIO Data In                (R)  (even in DS mode)
  4004C00h 1   GPIO Data Out               (W)
  4004C01h 1   GPIO Data Direction         (R/W)
  4004C02h 1   GPIO Interrupt Edge Select  (R/W)
  4004C03h 1   GPIO Interrupt Enable       (R/W)
  4004C04h 2   GPIO_WIFI                   (R/W)
```

ARM7 DSi CPU/Console ID (used as eMMC key)

```
  4004D00h 8   CPU/Console ID Code (64bit) (R)
  4004D08h 2   CPU/Console ID Flag (1bit)  (R)
```

ARM7 DSi Junk?

```
  8030200h 2   GBA area, accessed alongsides with SDIO port [4004A30h] (bug?)
```

Main Memory Control

```
2FFFFFEh 2   Main Memory Control (for 16MByte RAM chip)
DFFFFFEh 2   Main Memory Control (extra 16MByte RAM chip in debug version)
```

DSi Control Registers (SCFG)

4004000h - DSi9 - SCFG_A9ROM - ROM Status (R) [0000h]

```
0     ARM9 BIOS Upper 32K half of DSi BIOS (0=Enabled, 1=Disabled)
1     ARM9 BIOS for NDS Mode               (0=DSi BIOS, 1=NDS BIOS)
2-15  Unused (0)
16-31 Unspecified (0)
```

Possible values are:

```
00h  DSi ROM mapped at FFFFxxxxh, full 64K enabled (during bootstage 1 only)
01h  DSi ROM mapped at FFFFxxxxh, lower 32K only
03h  NDS ROM mapped at FFFFxxxxh (internal setting)
00h  NDS ROM mapped at FFFFxxxxh (visible setting due to SCFG_EXT.bit31=0)
```

Checking for A9ROM=01h is common for detecting if the console is a "DSi console running in DSi mode".

4004000h - DSi7 - SCFG_ROM - ROM Control (R/W, Set-Once)

```
0     ARM9 BIOS Upper 32K half of DSi BIOS (0=Enabled, 1=Disabled)
1     ARM9 BIOS for NDS Mode               (0=DSi BIOS, 1=NDS BIOS)
2-7   Unused (0)
8     ARM7 BIOS Upper 32K half of DSi BIOS (0=Enabled, 1=Disabled)
9     ARM7 BIOS for NDS Mode               (0=DSi BIOS, 1=NDS BIOS)
10    Access to Console ID registers       (0=Enabled, 1=Disabled) (4004Dxxh)
11-31 Unused (0)
```

Bits in this register can be set once (cannot be changed back from 1 to 0).
Don't change bit1 while executing IRQs or SWI functions on ARM9 side.
The System Menu sets bit10 shortly before starting any Cartridges or DSiware files (except System Base Tools) (for NDS mode, after having set bit10, it's also setting bit1 and bit9).

4004004h - DSi9 - SCFG_CLK - New Block Clock Control (R/W) [0084h]

```
0     ARM9 CPU Clock         (0=NITRO/67.03MHz, 1=TWL/134.06MHz) (TCM/Cache)
1     Teak DSP Block Clock   (0=Stop, 1=Run)
2     Camera Interface Clock (0=Stop, 1=Run)
3-6   Unused (0)
7     New Shared RAM Clock   (0=Stop, 1=Run)               (set via ARM7) (R)
8     Camera External Clock  (0=Disable, 1=Enable) ("outputs at 16.76MHz")
9-15  Unused (0)
16-31 See below (Port 4004006h, SCFG_RST)
```

Change ARM9 clock only from code within ITCM (and wait at least 8 cycles before accessing any non-ITCM memory).
Disable the corresponding modules before stopping their clocks.

4004004h - DSi7 - SCFG_CLK7 (R/W)

```
0     SD/MMC Clock      (0=Stop, 1=Run) (should be same as SCFG_EXT7.bit18)
1     Unknown/used      (0=Stop, 1=Run) (backlight goes off when cleared?)
2     Unknown/used      (0=Stop, 1=Run) (unknown effect?)
3-6   Unused (0)
7     New Shared RAM Clock   (0=Stop, 1=Run)
8     Touchscreen Clock (0=Stop, 1=Run) (needed for touchscr input)
9-15  Unused (0)
16-31 See below (Port 4004006h, SCFG_JTAG)
```

4004006h - DSi9 - SCFG_RST - New Block Reset (R/W) [0000h]

```
0     DSP Block Reset (0=Apply Reset, 1=Release Reset)
1-15  Unused (0)
```

4004006h - DSi7 - SCFG_JTAG - Debugger Control (R/W? or Write-ONCE-only?)

```
0     ARM7SEL (set when debugger can do ARM7 debugging)
1     CPU JTAG Enable
2-7   Unused (0)
8     DSP JTAG Enable
9-15  Unused (0)
```

Initialized as so: if SCFG_OP=2 then SCFG_JTAG=0102h, elseif SCFG_OP=1 then SCFG_JTAG=0103h, entrypoint=0, endif.

4004008h - DSi9 - SCFG_EXT - Extended Features (R/W) [8307F100h]

```
0     Revised ARM9 DMA Circuit       (0=NITRO, 1=Revised)
1     Revised Geometry Circuit       (0=NITRO, 1=Revised)
2     Revised Renderer Circuit       (0=NITRO, 1=Revised)
3     Revised 2D Engine Circuit      (0=NITRO, 1=Revised)
4     Revised Divider Circuit        (0=NITRO, 1=Revised)
5-6   Unused (0)
7     Revised Card Interface Circuit (0=NITRO, 1=Revised)
8     Extended ARM9 Interrupts       (0=NITRO, 1=Extended)
9-11  Unused (0)
12    Extended LCD Circuit           (0=NITRO, 1=Extended)
13    Extended VRAM Access           (0=NITRO, 1=Extended)
14-15 Main Memory RAM Limit (0..1=4MB/DS, 2=16MB/DSi, 3=32MB/DSiDebugger)
16    Access to New DMA Controller   (0=Disable, 1=Enable) (40041xxh)
17    Access to Camera Interface     (0=Disable, 1=Enable) (40042xxh)
18    Access to Teak DSP Block       (0=Disable, 1=Enable) (40043xxh)
19-23 Unused (0)
24    Access to 2nd NDS Cart Slot   (0=Disable, 1=Enable)  (set via ARM7) (R)
25    Access to New Shared WRAM     (0=Disable, 1=Enable)  (set via ARM7) (R)
26-30 Unused (0)
31    Access to SCFG/MBK registers  (0=Disable, 1=Enable) (4004000h-4004063h)
```

Bit24-25 are READ-ONLY (showing state set via ARM7 side). Note: Official specs for Bit24-25 are nonsense.
Default settings seem to be:

```
8307F100h for DSi firmware, DSi cartridges and DSiware
03000000h for NDS cartridges (and DSiware in NDS mode, eg. Pictochat)
```

Main RAM mapping depending on bit14-15:

```
Mode         2000000h-2FFFFFFh   C000000h-CFFFFFFh   D000000h-DFFFFFFh
4MB (0 or 1) 1st 4MB (+mirrors)  Zerofilled          Zerofilled
16MB (2)     1st 16MB            1st 16MB (mirror)   1st 16MB (mirror)
32MB (3)     1st 16MB            1st 16MB (mirror)   Open bus (or 2nd 16MB)
```

DSi9 SCFG_EXT.bit14-15 affect the Main RAM mapping on <both> ARM9 and ARM7 side (that, at least AFTER games have been booted, however, there's a special case DURING boot process: For NDS games, the firmware switches to 4MB mode on ARM9 side, whilst ARM7 is still relocating memory from the 16MB area at the same time - unknown how that is working exactly, maybe ARM7 isn't affected by ARM9 SCFG_EXT setting until ARM7 has configured/disabled its own SCFG_EXT register).
The 32MB mode requires an extra RAM chip (present in DSi debug version only; DSi retail consoles return 16bit open bus values instead of extra memory). RAM Size/Openbus detection is conventionally done by trying to read/write a BYTE at [0DFFFFFAh].

4004008h - DSi7 - SCFG_EXT7 - Extended Features (R/W)

```
0     Revised ARM7 DMA Circuit       (0=NITRO, 1=Revised)
1     Revised Sound DMA              (0=NITRO, 1=Revised)
2     Revised Sound                  (0=NITRO, 1=Revised)
3-6   Unused (0)
7     Revised Card Interface Circuit (0=NITRO, 1=Revised) (set via ARM9) (R)
8     Extended ARM7 Interrupts      (0=NITRO, 1=Extended) (4000218h)
9     Extended SPI Clock (8MHz)     (0=NITRO, 1=Extended) (40001C0h)
10    Extended Sound DMA        ?   (0=NITRO, 1=Extended) (?)
11    Undocumented/Unknown      ??  (0=NITRO, 1=Extended) (?)
12    Extended LCD Circuit          (0=NITRO, 1=Extended) (set via ARM9) (R)
13    Extended VRAM Access          (0=NITRO, 1=Extended) (set via ARM9) (R)
```

```
  14-15 Main Memory RAM Limit    (0..1=4MB, 2=16MB, 3=32MB) (set via ARM9) (R)
  16    Access to New DMA Controller  (0=Disable, 1=Enable) (40041xxh)
  17    Access to AES Unit            (0=Disable, 1=Enable) (40044xxh)
  18    Access to SD/MMC registers    (0=Disable, 1=Enable) (40048xxh-40049xxh)
  19    Access to SDIO Wifi registers (0=Disable, 1=Enable) (4004Axxh-4004Bxxh)
  20    Access to Microphone regs     (0=Disable, 1=Enable) (40046xxh)
  21    Access to SNDEXCNT register   (0=Disable, 1=Enable) (40047xxh)
  22    Access to I2C registers       (0=Disable, 1=Enable) (40045xxh)
  23    Access to GPIO registers      (0=Disable, 1=Enable) (4004Cxxh)
  24    Access to 2nd NDS Cart Slot   (0=Disable, 1=Enable) (40021xxh)
  25    Access to New Shared WRAM     (0=Disable, 1=Enable) (3xxxxxxh)
  26-27 Unused (0)
  28    Undocumented/Unknown          (0=???, 1=Normal)     (?)
  29-30 Unused (0)
  31    Access to SCFG/MBK registers  (0=Disable, 1=Enable) (4004000h-4004063h)
```
Bit7,12-15 are READ-ONLY (showing state set via ARM9 side).
Default settings seem to be:
```
  93FFFB06h for DSi Firmware (Bootcode and SysMenu/Launcher)
  13FFFB06h for DSiware (eg. SysSettings, Flipnote, PaperPlane)
  13FBFB06h for DSi Cartridges (eg. System Flaw)           (bit18=0=sdmmc off)
  12A03000h for NDS cartridges (and DSiware in NDS mode, eg. Pictochat)
```
Bits 0,1,2,10,18,31 are taken from carthdr[1B8h].

4004010h - DSi9 - SCFG_MC - NDS Slot Memory Card Interface Status (R)
4004010h - DSi7 - SCFG_MC - NDS Slot Memory Card Interface Control (R/W)
```
  0     1st NDS Slot Game Cartridge (0=Inserted, 1=Ejected)              (R)
  1     1st NDS Slot Unknown/Unused (0)
  2-3   1st NDS Slot Power State (0=Off, 1=On+Reset, 2=On, 3=RequestOff) (R/W)
  4     2nd NDS Slot Game Cartridge (always 1=Ejected) ;\DSi             (R)
  5     2nd NDS Slot Unknown/Unused (0)                ; prototype
  6-7   2nd NDS Slot Power State    (always 0=Off)     ;/relict          (R/W)
  8-14  Unknown/Undocumented (0)
  15    Swap NDS Slots (0=Normal, 1=Swap)                               (R/W)
  16-31 ARM7: See Port 4004012h, ARM9: Unspecified (0)
```
Note: Additionally, the NDS slot Reset pin can be toggled (via ROMCTRL.Bit29; that bit is writeable on ARM7 side on DSi; which wasn't supported on NDS).
Power state values:
```
  0=Power is Off
  1=Power On and force Reset (shall be MANUALLY changed to state=2)
  2=Power On
  3=Request Power Off (will be AUTOMATICALLY changed to state=0)
```
cart_power_on: (official/insane 1+10+27+120ms, but also works with 1+1+0+1ms)
```
  wait until state<>3                   ;wait if pwr off busy
  exit if state<>0 AND no_reset_wanted  ;exit if already on & no reset wanted
  wait 1ms, then set state=1            ;pwr on & force reset
  wait 10ms, then set state=2           ;pwr on normal state   ;better: 1ms
  wait 27ms, then set ROMCTRL=20000000h ;release reset pin     ;better: 0ms
  wait 120ms (or 270ms on 3DS)          ;more insane delay?    ;better: 1ms
```
cart_power_off: (official 150ms, when using default [4004014h]=264Ch)
```
  wait until state<>3                   ;wait if pwr off busy
  exit if state<>2                      ;exit if already off
  set state=3                           ;request pwr off
  exit unless you want to know when below pointless delay has ellapsed
  wait until state=0    ;default=150ms  ;wait until pwr off    ;better: skip
```
Power Off is also done automatically by hardware when ejecting the cartridge.
The Power On sequence does reset ROMCTRL.bit29=0 (reset signal).
Bit15 swaps ports 40001A0h-40001BFh and 4100010h with 40021A0h-40021BFh? and 4102010h?, the primary purpose is mapping the 2nd Slot to the 4xx0xxxh registers (for running carts in 2nd slot in NDS mode; which of course doesn't work because the 2nd slot connector isn't installed), theoretically it would also allow to access the 1st slot via 4xx2xxxh registers (however, that doesn't seem to be fully implemented, cart reading does merely reply FFh's (cart inserted) or 00h's (no cart)). 4102010h can be read by manually polling DRQ in

40021A4h.bit23, and probably by NDMA (but not by old DMA which has no known DRQ mode for 2nd slot).

4004012h - DSi7 - SCFG_CARD_INSERT_DELAY (usually 1988h = 100ms) (R/W)
4004014h - DSi7 - SCFG_CARD_PWROFF_DELAY (usually 264Ch = 150ms) (R/W)

```
  0-15  Delay in 400h cycle units (at 67.027964MHz)  ;max FFFFh=ca. 1 second
```

Usually set to 1988h/264Ch by firmware. Power up default is FFFFh/FFFFh, with that setting it takes about 1 second to sense inserted carts (after power-up, or after cart-insert).
Insert delay defines the time until SCFG_MC.bit0 reacts upon cart insert, and until cart access works (cart eject works instantly regardless of the delay setting). The 3DS bootrom uses 051Dh=20ms, which may be good to avoid switch bounce. The DSi/3DS firmwares use 1988h=100ms, which might be reasonable if the switch triggers too early.
Power Off delay defines how fast SCGF_MC.bit2-3 are changing from state=3 (request power off) to state=0 (power off). The 3DS bootrom uses 051Dh=20ms, the DSi/3DS firmwares use 264Ch=150ms, which is both kinda pointless. If the VCC pin is kept powered during that time (?) then the delay might help to finish FLASH writes (which would make sense only if there was data written to cartridge).

4004020h - DSi7 - SCFG_WL - Wireless Disable (R/W)

```
  0     OFFB, related to Wifi Enable flag from TWLCFGn.dat files?
  1-15  Unknown/unused (0)
```

Unknown what that does. SDIO Wifi seems to work regardless of that bit.

4004024h - DSi7 - SCFG_OP - Debugger Type (R)

```
  0-1   Debug Hardware Type (0=Retail, other=debug variants)
  2-3   Unknown/unused (0)
  4     Unknown (maybe used, since it isn't masked & copied to RAM)
  5-15  Unknown/unused (0)
```

Changing this register would theoretically allow to install the debug firmware on retail consoles, however, it's unknown where bit0-1 come from (possibly from some external FLASH memory, internal PROM in the TWL CPU, solder pads underneath of the TWL CPU, or even from a physically different TWL CPU chip version).

DSi XpertTeak (DSP)

The DSi includes an XpertTeak Digital Signal Processor (DSP); which is consisting of a TeakLite II processor, plus some "expert" features (like DMA support). The thing appears to be intended for audio/video decoding, but it's left unused by most DSi games. However, it's used by the "Nintendo DSi Sound" and "Nintendo Zone" system utilities, and by the "Cooking Coach" cartridge.

DSi Teak Misc

XpertTeak I/O Registers (on ARM side)
DSi Teak I/O Ports (on ARM9 Side)

XpertTeak I/O Registers (on Teak side)
DSi Teak MMIO - Register Summary
DSi Teak MMIO[8000h] - Misc Registers (JAM/GLUE)
DSi Teak MMIO[8020h] - Timers (TMR)
DSi Teak MMIO[8050h] - Serial Port (SIO)
DSi Teak MMIO[8060h] - Debug (OCEM, On-chip Emulation Module)
DSi Teak MMIO[8080h] - PLL and Power (PMU, Power Management Unit)
DSi Teak MMIO[80C0h] - Host Port Interface (APBP aka HPI)
DSi Teak MMIO[80E0h] - AHBM - Advanced High Performance Bus Master
DSi Teak MMIO[8100h] - Memory Interface Unit (MIU)
DSi Teak MMIO[8140h] - Code Replacement Unit (CRU)
DSi Teak MMIO[8180h] - Direct Memory Access (DMA)

TeakLite II Processor

DSi Teak Misc

Teak Instruction Set References
There aren't any official references for the Teak instruction set. However, there's one document that has leaked into internet (plus some docs for older Oak instruction set):
```
TeakLite Architecture Specification Revision 4.41 (DSP Group Inc.)
OakDSPCore Technical Manuals for CWDSP1640 or CWDSP167x (LSI Logic)
OakDSPCore DSP Subsystem AT75C (Atmel)
```
TeakLite II supports lots of additional opcodes, the only available info has leaked in form of .DLLs which were (apparently by mistake) bundled with a specific RVDS release version:
```
TeakLite II disassembler dll in RVDS (RealView Developer Suite) 4.0 Pro
```
There's no known way to use the RVDS disassembler GUI to decipher Teak binaries. However, Normmatt found a way to get the .DLLs to disassemble code manually (via LoadLibrary and GetProcAddr), which in turn allowed to disassemble all possible 65536 combinations for all opcodes.
BUG: That RVDS tool disassembles "R0425" operands to "r6" (instead of "r5"), that's definetely wrong for the old "movr" opcode (TeakLite I didn't support any "r6" register at all), and, when looking at disassembled code, it does also look wrong for newer TL2 opcodes.
Actually, the registers CAN be redefined via ar0/ar1/arp0/arp1/arp2/arp3, so it isn't actually a bug.

Teak MMIO Registers
There aren't official docs, but wwylele has discovered a debugger at that contains definitions for most of the MMIO bits:
```
searching for "teak" in the "search for chip" box on "www.lauterbach.com"
should lead to this file: trace32_ceva-teak_r_2019_02_000108303_win64.zip
```
The zip contains a "perxteak.per" file that contains the MMIO bit definitions (in txt format). The definitions for OCEM do accidentally refer to an older version (without 18bit addresses). Definitions for GLUE, ICU, APBP, DMA, CRU, GPIO, and BTDMP are missing (whereof a similar/older ICU variant is defined in "peroak.per" in the same zip).

Teak COFF Files
The DSi Sound utility contains a COFF file called "aac.a" (inside of its nitro filesystem), and aside from the binary, the file is also including a COFF symbol table with labels in ASCII format.

Teak Undefined Opcodes
There are several "Undefined" opcodes: Any opcodes that have no instruction assigned in the opcode encoding table (or that are excplicitely assigned as "undefined" in the table). Opcodes with invalid parameters (eg. ArArp set to 6..7).
Some opcodes are also having "Unused" operand bits; these bits should be usually zero (nonzero would supposedly mirror to the same instruction, but one shouldn't do that).
Moreover, there are various special cases saying that certain opcodes may not be used with certain registers, eg. "addh" shall not be used with operands Ax,Bx,p (with unknown results when violating that rule).

Teak Memory

Memory is addressed in 16bit WORD units (not in bytes) with separate Instruction and Data busses. Before starting the Teak, store the Teak program code in New Shared WRAM, and then map that memory to Teak side via MBK registers:
DSi New Shared WRAM (for ARM7, ARM9, DSP)
At Teak side, 16bit is the smallest addressable unit (so there's no "byte-order" on Teak side - however, 16bit values should be stored in little endian format on ARM side).
Confusingly, the "movpdw" opcode is doing a 32bit read with two 16bit words ordered in big-endian (and, on ARM side, byte-fractions ordered in little-endian). There are a few more opcodes that can read/write 32bits, with optional address increment/decrement for the 2nd word, so endianness is selectable in that cases; it's also common to use the SAME address for both words (probably intended for scaling a 16bit memory value to 32bits).

Teak Code Memory
TeakLite II supports 18bit program memory addressing (unlike Teak/Oak which supported only 16bit addresses). The 18bits allow to address max 256Kwords (=512Kbytes) of code, whereas, the DSi can map only half that much memory to the DSP (ie. max 256Kbytes code).
Call/Ret are always pushing/popping 32bit return addresses (even when doing "near" calls within the same 16bit page).

Teak Data Memory (RAM and Memory Mapped I/O)
Teak Data Memory is addressed via 16bit address bus (via registers r0..r7), allowing to access max 64Kwords (2Kwords of MMIO, plus 62Kwords of RAM). The memory is divided into three sections (X/Z/Y-spaces), the size/location of that sections can be changed via Port 8114h (in 1Kword units), and alongsides, the MMIO base can be adjusted via Port 811Eh. The default areas are:

```
  0000h..7FFFh  X Space (for RAM, with 1-stage write-buffer)     ;min zero
  8000h..87FFh  Z Space (for Memory-mapped I/O, no write-buffer) ;min zero
  8800h..FFFFh  Y Space (for RAM, with 1-stage write-buffer))    ;min 1Kword
```

Confusingly, the DSi Sound utility is mapping 128Kwords of RAM as Teak Data memory, but it's unknown how to access all of that memory. The CPU opcodes, CPU registers, and MMIO registers don't seem to allow to access more than 64Kwords of Data. Maybe the extra memory is accessible via DMA, and maybe the CPU's [r7+imm16] operands might allow to exceed the 64Kbyte range (though they might as well wrap within 64Kbyte range, actually that's more likely, especially for "signed" immediates).

Teak Call Conventions (as done in "aac.a" from DSi Sound)
Functions are called with up to four parameters in a0,a1,b0,b1 (or a0l, a1l, b0l, b1l when needing only 16bits). Any further parameters are pushed on stack before the function call (and are deallocated via "rets Imm8" opcode upon return). Register r7 is often used as stack frame (for accessing pushed incoming parameters & locally allocated variables).
Functions may smash a0, a1, b0, b1. A return value (if any) is stored in a0 (or a0l). All other registers like r0..r7, sv, etc. should be left unchanged (or pushed/popped when needed).

Teak Speed
Cycles per opcode are defined in the TeakLite document (not covering TeakLite II opcodes though). Most instructions (even Multiply opcodes) can complete in a single clock cycle. The main bottleneck appear to be memory access cycles: Code and Data memory can be accessed in parallel, so the overall rule would be:

```
  NumCycles = max(NumberOfOpcodeWords, NumberOfDataReadsWrites)
```

Some exceptions with extra cycles are opcodes that are changing PC, or that do read/write program memory (movd and movp). Opcodes exp, max, maxd, min are having restrictions saying that the result may not be used by the "following instruction".
The overall clock speed in the DSi is unknown; some years ago somebody seems to have claimed it to be around 130MHz, but it's unclear where that info came from.

```
  Teak actual CPU clock(s) are...
  134.055928MHz (aka 134MHz)       <-- for Timer 0, SIO, DMA (no "waitstates")
  107.244742MHz (aka 134MHz/1.25)  <-- for Timer 1, CPU    (with "waitstates")
```

The ARM9 can access WRAM at 33MHz, whilst Teak seems to be capable of fetching 16bit opcodes and 16bit data at 107MHz from the same WRAM (apparently without even using a cache).

DSi Teak I/O Ports (on ARM9 Side)

4004300h - DSi9 - DSP_PDATA - DSP Transfer Data Read FIFO (R)

```
0-15  Data (one stage of the 16-stage Read FIFO)
```

4004300h - DSi9 - DSP_PDATA - DSP Transfer Data Write FIFO (W)

```
0-15  Data (one stage of the 16-stage Write FIFO)
```

4004304h - DSi9 - DSP_PADR - DSP Transfer Address (W)

```
0-15  Lower 16bit of Address in DSP Memory (in 16bit units)
```

For MMIO, only lower 10bit are used (ie. MMIO[8000h-87FFh] can be accessed via 8000h-87FFh or 0000h-07FFh or other mirrors).
For Data, the 16bit address allows to access 64K-words.
For Code, the 16bit address allows to access 64K-words, but Code works only in write-direction?
For Data/Code, the upper 16bit of Address must be configured on DSP side, in DMA channel 0 registers:

```
MMIO[81BEh]   - DMA Select Channel (must be 0 for below DMA 0 regisrers)
MMIO[81C2h:0] - DMA Channel 0: Source Address, bit16-31 (R/W)
MMIO[81C6h:0] - DMA Channel 0: Destination Address, bit16-31 (R/W)
```

The Auto-increment flag in DSP_PCFG.bit1 will increment the 16bit address after each DSP_PDATA unit (but doesn't seem to increment the upper bits after address xxFFFFh).

4004308h - DSi9 - DSP_PCFG - DSP Configuration (R/W)

```
0     DSP Reset (0=Release, 1=Reset) ;should be held "1" for 8 DSP clks
1     DSP Transfer Address Auto-Increment  (0=Off, 1=On)
2-3   DSP Read Data Length (0=1 word, 1=8 words, 2=16 words, 3=FreeRun)
4     DSP Read Start Flag (mem transfer via Read FIFO) (1=Start)
5     Interrupt Enable Read FIFO Full      (0=Off, 1=On)
6     Interrupt Enable Read FIFO Not-Empty (0=Off, 1=On)
7     Interrupt Enable Write FIFO Full     (0=Off, 1=On)
8     Interrupt Enable Write FIFO Empty    (0=Off, 1=On)
9     Interrupt Enable Reply Register 0    (0=Off, 1=On)
10    Interrupt Enable Reply Register 1    (0=Off, 1=On)
11    Interrupt Enable Reply Register 2    (0=Off, 1=On)
12-15 DSP Memory Transfer (0=DataMem, 1=MMIO Register, 5=ProgramMem)
```

400430Ch - DSi9 - DSP_PSTS - DSP Status (R)

```
0     Read Transfer Underway Flag    (0=No, 1=Yes/From DSP Memory)
1     Write Transfer Underway Flag   (0=No, 1=Yes/To DSP Memory)
2     Peripheral Reset Flag          (0=No/Ready, 1=Reset/Busy)
3-4   Unused (0)
5     Read FIFO Full Flag            (0=No, 1=Yes/Full)
6     Read FIFO Not-Empty Flag       (0=No, 1=Yes, ARM9 may read PDATA)
7     Write FIFO Full Flag           (0=No, 1=Yes/Full)
8     Write FIFO Empty Flag          (0=No, 1=Yes/Empty)
9     Semaphore IRQ Flag             (0=None, 1=IRQ)
10    Reply Register 0 Update Flag   (0=Was Written by DSP, 1=No)
11    Reply Register 1 Update Flag   (0=Was Written by DSP, 1=No)
12    Reply Register 2 Update Flag   (0=Was Written by DSP, 1=No)
13    Command Register 0 Read Flag   (0=Was Read by DSP, 1=No)
14    Command Register 1 Read Flag   (0=Was Read by DSP, 1=No)
15    Command Register 2 Read Flag   (0=Was Read by DSP, 1=No)
```

Unknown if/when bit10-15 get reset... maybe after reading the status... or when reading a reply or writing a new command?

4004310h - DSi9 - DSP_PSEM - ARM9-to-DSP Semaphore (R/W)

```
  0-15  ARM9-to-DSP Semaphore 0..15 Flags (0=Off, 1=On)
```
Reportedly these flags are sent in ARM9-to-DSP direction (=seems correct).
Confusingly, the other DSP_Pxxx registers are for opposite direction?

4004314h - DSi9 - DSP_PMASK - DSP-to-ARM9 Semaphore Mask (R/W)
```
  0-15  DSP-to-ARM9 Semaphore 0..15 Interrupt Disable (0=Enable, 1=Disable)
```

4004318h - DSi9 - DSP_PCLEAR - DSP-to-ARM9 Semaphore Clear (W)
```
  0-15  DSP-to-ARM9 Semaphore 0..15 Clear (0=No Change, 1=Clear/Ack)
```
Reportedly clears bits in DSP_PSEM/4004310h. [that's probably nonsense, clearing bits in DSP_SEM/400431Ch would make more sense?]

400431Ch - DSi9 - DSP_SEM - DSP-to-ARM9 Semaphore Data (R)
```
  0-15  DSP-to-ARM9 Semaphore 0..15 Flags (0=Off, 1=On)
```
Reportedly these flags are received in DSP-to-ARM9 direction.

4004320h - DSi9 - DSP_CMD0 - DSP Command Register 0 (R/W) (ARM9 to DSP)
4004328h - DSi9 - DSP_CMD1 - DSP Command Register 1 (R/W) (ARM9 to DSP)
4004330h - DSi9 - DSP_CMD2 - DSP Command Register 2 (R/W) (ARM9 to DSP)
```
  0-15  Command/Data to DSP
```

4004324h - DSi9 - DSP_REP0 - DSP Reply Register 0 (R) (DSP to ARM9)
400432Ch - DSi9 - DSP_REP1 - DSP Reply Register 1 (R) (DSP to ARM9)
4004334h - DSi9 - DSP_REP2 - DSP Reply Register 2 (R) (DSP to ARM9)
```
  0-15  Reply/Data from DSP
```

Further Teak related registers
SCFG_CLK, SCFG_RST, SCFG_EXT registers, MBK registers, and SNDEXCNT register.
DSi Control Registers (SCFG)
DSi New Shared WRAM (for ARM7, ARM9, DSP)
DSi Microphone and SoundExt
And, for the final audio output and microphone input,
DSi Touchscreen/Sound Controller

DSi Teak MMIO - Register Summary

I/O ports are mapped in "data memory" at 8000h-87FFh (2Kwords). All ports are 16bit wide, located at even word addresses (words at odd addresses seem to be always 0000h, except, there ARE some odd ports used at 806xh).

8000h Mirrors
```
  8000h..8002h 3300 3300 3300 R   Mirror of Port 80D6h
```
8004h JAM
```
  8004h        0000 0000 87FF R/W JAM Unknown
  8006h        ?    ?    ?    ??  JAM Unknown/Crash, DANGER (crashes on read)
  8008h..800Eh 3300 3300 3300 R   Mirror of Port 80D6h
```
8010h GLUE
```
  8010h        0000 0000 0003 R/W GLUE CFG0
  8012h        0000 0000 0003 R/W GLUE Unknown 2bit
  8014h        0000 0000 FFFF R/W GLUE Unknown 16bit
  8016h        0000 0000 0000 R   GLUE Unknown (DSi=0000h, New3DS=00BAh)
  8018h        0000 0000 BDEF R/W GLUE Whatever Parity/Shuffle
  801Ah        C902 C902 C902 R   GLUE Chip config ID (for xpert_offsets_tbl)
  801Ch..801Eh 0003 0003 0003 R   Mirror of port 8010h
```
8020h Timers

```
  8020h        0000 0000 7xDF R/W Timer 0 Control (bit11=DANGER)      ;\
  8022h        0000 0000 0000 W   Timer 0 Trigger Event/Watchdog      ;
  8024h        0000 0000 FFFF R/W Timer 0 Reload value, bit0-15       ; Timer 0
  8026h        0000 0000 FFFF R/W Timer 0 Reload value, bit16-31      ;
  8028h        0000 0000 0000 R   Timer 0 Counter value, bit0-15      ;
  802Ah        0000 0000 0000 R   Timer 0 Counter value, bit16-31     ;
  802Ch        0000 0000 FFFF R/W Timer 0 PWM Reload value, bit0-15   ;
  802Eh        0000 0000 FFFF R/W Timer 0 PWM Reload value, bit16-31  ;/
  8030h        0200 0200 7xDF R/W Timer 1 Control (bit11=DANGER)      ;\
  8032h        0000 0000 0000 W   Timer 1 Trigger Event/Watchdog      ;
  8034h        0000 0000 FFFF R/W Timer 1 Reload value, bit0-15       ; Timer 1
  8036h        0000 0000 FFFF R/W Timer 1 Reload value, bit16-31      ;
  8038h        0000 0000 0000 R   Timer 1 Counter value, bit0-15      ;
  803Ah        0000 0000 0000 R   Timer 1 Counter value, bit16-31     ;
  803Ch        0000 0000 FFFF R/W Timer 1 PWM Reload value, bit0-15   ;
  803Eh        0000 0000 FFFF R/W Timer 1 PWM Reload value, bit16-31  ;/
  8040h..804Eh 3300 3300 3300 R   Mirror of Port 80D6h
```

8050h Serial Port (SIO)

```
  8050h        7000 0000 F03F R/W SIO Control
  8052h        0000 0000 7F7F R/W SIO Clock Divider
  8054h        0000 0000 0000 R+W SIO Data (R) and (W)
  8056h        0000 0000 0001 R/W SIO Enable
  8058h        0000 0000 0000 R   SIO Status
  805Ah..805Eh F03F F03F F03F R   Mirror of port 8050h
```

8060h Debug (OCEM, On-chip Emulation Module)

```
  8060h        0105 0105 0105 R   OCEM Program Flow Trace Buffer, bit0-15
  8061h        0000 0000 0000 R   OCEM Program Flow Trace Buffer, bit16-31
  8062h        FFFF 0000 FFFF R/W OCEM Program Break Address 1, bit0-15
  8063h        0F03 0000 0F03 R/W OCEM Program Break Address 1, bit16-31
  8064h        FFFF 0000 FFFF R/W OCEM Program Break Address 2, bit0-15
  8065h        0F03 0000 0F03 R/W OCEM Program Break Address 2, bit16-31
  8066h        FFFF 0000 FFFF R/W OCEM Program Break Address 3, bit0-15
  8067h        0F03 0000 0F03 R/W OCEM Program Break Address 3, bit16-31
  8068h        00FF 0000 00FF R/W OCEM Program Break Counter 1
  8069h        00FF 0000 00FF R/W OCEM Program Break Counter 2
  806Ah        00FF 0000 00FF R/W OCEM Program Break Counter 3
  806Bh        FFFF 0000 FFFF R/W OCEM Data Break Mask
  806Ch        FFFF 0000 FFFF R/W OCEM Data Break Address
  806Dh        0000 0000      R/W OCEM Breakpoint Enable Flags (DANGER)
  806Eh        3001 0000 FFFF R/W OCEM Mode/Indication?
  806Fh        0000 0000 BFFF R/W OCEM Breakpoint Status Flags
  8070h        0000 0000 0001 R/W OCEM Program Flow Trace Update Disable
  8072h        0000 0000 FFFF R/W Unknown 16bit?
  8074h        C000 C000 C000 R   OCEM Boot/Debug Mode
  8076h..807Eh 0105 0105 0105 R   Mirror of port 8060h
```

8080h PLL and Power (PMU, Power Management Unit)

```
  8080h        C00E 0000 FFFF R/W PMU PLL Multiplier
  8082h        0001 0000 0001 R/W PMU PLL Power-on config
  8084h        8000           R/W PMU PLL Divider/Bypass (DANGER)
  8086h        0000           R/W PMU Wake/Shutdown Module(s)
  8088h        0000 0000 07BF R/W PMU Recover Module(s) on interrupt 0
  808Ah        0000 0000 07BF R/W PMU Recover Module(s) on interrupt 1
  808Ch        0000 0000 07BF R/W PMU Recover Module(s) on interrupt 2
  808Eh        0000 0000 07BF R/W PMU Recover Module(s) on vectored interrupt
  8090h        0000 0000 06BF R/W PMU Recover Module(s) on Timer 0 (no bit8)
  8092h        0000 0000 05BF R/W PMU Recover Module(s) on Timer 1 (no bit9)
  8094h        0000 0000 07BF R/W PMU Recover Module(s) on NMI
  8096h        0000 0000 0002 R/W PMU Recover DMA on external signal (bit1)
  8098h        0000 0000 0302 R/W PMU Breakpoint mask module(s) (bit1,8,9 only)
  809Ah        0000 0000 0003 R/W PMU Wake/Shutdown BTDMP(s)
  809Ch        0000 0000 0003 R/W PMU Recover BTDMP(s) on interrupt 0
  809Eh        0000 0000 0003 R/W PMU Recover BTDMP(s) on interrupt 1
  80A0h        0000 0000 0003 R/W PMU Recover BTDMP(s) on interrupt 2
  80A2h        0000 0000 0003 R/W PMU Recover BTDMP(s) on vectored interrupt
```

```
  80A4h        0000 0000 0003 R/W PMU Recover BTDMP(s) on Timer 0
  80A6h        0000 0000 0003 R/W PMU Recover BTDMP(s) on Timer 1
  80A8h        0000 0000 0003 R/W PMU Recover BTDMP(s) on NMI (undoc?)
  80AAh        0000 0000 FFFF R/W Unknown 16bit
  80ACh        0000 0000 FFFF R/W Unknown 16bit
  80AEh        0000 0000 FFFF R/W Unknown 16bit
  80B0h..80BEh FFFF FFFF FFFF R   Mirror of port 8080h
```

80C0h APBP (aka HPI Host Port Interface)

```
  80C0h        xxxx xxxx xxxx R/W APBP DSP-to-ARM Reply 0
  80C2h        4300 4300 4300 R   APBP ARM-to-DSP Command 0
  80C4h        0000 0000 FFFF R/W APBP DSP-to-ARM Reply 1
  80C6h        3123 3123 3123 R   APBP ARM-to-DSP Command 1
  80C8h        0000 0000 FFFF R/W APBP DSP-to-ARM Reply 2
  80CAh        3223 3223 3223 R   APBP ARM-to-DSP Command 2
  80CCh        0000 0000 FFFF R/W APBP DSP-to-ARM Semaphore Set Flags
  80CEh        0000           R/W APBP ARM-to-DSP Semaphore Interrupt Mask
  80D0h        0000           W?  APBP ARM-to-DSP Semaphore Ack Flags
  80D2h        AFFE AFFE AFFE R   APBP ARM-to-DSP Semaphore Get Flags
  80D4h        0000           R/W APBP Control (DANGER: can crash cpu)
  80D6h        03C0 03C0 03C0 R   APBP DSP-side Status
  80D8h        3B00 3B00 3B00 R   APBP ARM-side Status (mirror of 400430Ch)
  80DAh..80DEh 0000 0000 0000 R   Fixed 0000h
```

80E0h Bus Config for DMA to external ARM-bus (AHBM)

```
  80E0h        0000 0000 0000 R   AHBM Status
  80E2h+N*6    0000 0000 0FBF R/W AHBM Channel 0..2 Configure Burst/Data
  80E4h+N*6    0000 0000 03FF R/W AHBM Channel 0..2 Configure Whatever
  80E6h+N*6    0000 0000 00FF R/W AHBM Channel 0..2 Configure DMA
  80F4h        0000 0000 FC00 R/W Unknown 6bit? bit10-15 are used
  80F6h        0000 0000 0000 R?  AHBM Internal FIFO (R) and maybe also (W?)
  80F8h        0000 0000 0000 R?  Unknown always zero?
  80FAh        0000 0000 FFFF R/W Read/write-able(!) mirror of MMIO[80FCh]
  80FCh        FFFF 0000 FFFF R/W Unknown 16bit?
  80FEh        0000 0000 FFFF R/W Unknown 16bit?
```

8100h MIU (memory limits for X-,Y-,Z-Space, and Memory Mapped I/O base)

```
  8100h        FFFF 0000 FFFF R/W MIU Waitstate Settings, bit0-15
  8102h        0FFF 0000 0FFF R/W MIU Waitstate Settings, bit16-31
  8104h        0000 0000 FFFF R/W MIU Waitstate Area Z0
  8106h        0000 0000 FFFF R/W MIU Waitstate Area Z1
  8108h        0000 0000 FFFF R/W MIU Waitstate Area Z2
  810Ah        0000 0000 FFFF R/W MIU Waitstate Area Z3
  810Ch        0014 0014 0014 R   Mirror of port 811Ah
  810Eh        0000 0000 FFFF R/W MIU X Page (16bit) (or unused)
  8110h        0000 0000 00FF R/W MIU Y Page (8bit)  (or unused)
  8112h        0000           R/W MIU Z Page (16bit) (or absolute page)(DANGER)
  8114h        1E20           R/W MIU X/Y Page Size for Page 0 (or all pages)
  8116h        1E20 0100 403F R/W MIU X/Y Page Size for Page 1 (or unused)
  8118h        1E20 0100 403F R/W MIU X/Y Page Size for Off-chip (or unused)
  811Ah        0014 00x4      R/W MIU Config for Misc stuff (DANGER)
  811Ch        0004 0000 007F R/W MIU Config for Program Page and Download Mem
  811Eh        8000           R/W MIU Base Address for MMIO Registers (DANGER)
  8120h        0000 0000 000F R/W MIU Observability Mode
  8122h        0000 0000 007F R/W MIU Pin Config?
  8124h..813Eh 0014 0014 0014 R   Mirror of port 811Ah
```

8140h Patch (CRU, Code Replacement Unit)

```
  8140h+N*4    0000 0000 FFFF R/W CRU Entry 0..14 Offset, bit0-15
  8142h+N*4    0000 0000 803F R/W CRU Entry 0..14 Offset, bit16-31
  817Ch        0000 0000 FFFF R/W CRU Entry 15 Offset, bit0-15  ;\with control
  817Eh        0000 0000 C03F R/W CRU Entry 15 Offset, bit16-31 ;/status bits
```

8180h DMA (eight channels, port 81C0h..81Exh are bank-switched via 81BEh)

```
  8180h        0000 0000 0000 R   DMA Internal: Channel Size0 Busy or so?
  8182h        0000 0000 0000 R   DMA Internal: Channel Size1 Busy or so?
  8184h        0001 0000 00FF R/W DMA Channel Start Flags (1=Start/Busy)
  8186h        0000 0000 00FF R/W DMA Channel Pause Flags (1=Pause)
  8188h        0000 0000 0000 R   DMA Channel End Flags for Size0
```

```
  818Ah        0000 0000 0000 R   DMA Channel End Flags for Size1
  818Ch        0000 0000 0000 R   DMA Channel End Flags for Size2 (all done)
  818Eh        3210 0000 7777 R/W DMA Whatever Slot Config, bit0-15
  8190h        7654 0000 7777 R/W DMA Whatever Slot Config, bit16-31
  8192h        0000 0000 7C03 R/W Unknown, R/W mask 7C03h
  8194h        0000 0000 0000 R   DMA Internal: contains SRC_ADDR_L after DMA
  8196h        0000 0000 0000 R   DMA Internal: contains DST_ADDR_L after DMA
  8198h..81B4h 0000 0000 0000 R   Fixed 0000h
  81B6h        0000 0000 FFFF R/W Unknown, 16bit
  81B8h        0000 0000 FFFF R/W Unknown, 16bit
  81BAh        0000 0000 FFFF R/W Unknown, 16bit
  81BCh        0000 0000 FFFF R/W Unknown, 16bit
  81BEh        0000 0000 0007 R/W DMA Select Channel (bank for 81C0h-81Exh)
  81C0h:0..7   0000 0000 FFFF R/W DMA Channel: Source Address, bit0-15
  81C2h:0..7   0000 0000 FFFF R/W DMA Channel: Source Address, bit16-31
  81C4h:0..7   0000 0000 FFFF R/W DMA Channel: Destination Address, bit0-15
  81C6h:0..7   0000 0000 FFFF R/W DMA Channel: Destination Address, bit16-31
  81C8h:0..7   FFFF 0001 FFFF R/W DMA Channel: Size0 (usually total len)
  81CAh:0..7   0001 0001 FFFF R/W DMA Channel: Size1 (usually 1)
  81CCh:0..7   0001 0001 FFFF R/W DMA Channel: Size2 (usually 1)
  81CEh:0..7   0001 0000 FFFF R/W DMA Channel: Source Step0      ;-2,4,2,1
  81D0h:0..7   0001 0000 FFFF R/W DMA Channel: Source Step1      ;-4,2,2,1
  81D2h:0..7   0001 0000 FFFF R/W DMA Channel: Source Step2      ;-2,4,0,1
  81D4h:0..7   0001 0000 FFFF R/W DMA Channel: Destination Step0 ;-4,2,0,1
  81D6h:0..7   0001 0000 FFFF R/W DMA Channel: Destination Step1 ;-0,0,0,1
  81D8h:0..7   0001 0000 FFFF R/W DMA Channel: Destination Step2 ;-0,0,0,1
  81DAh:0..7   F200 0000 F7FF R/W DMA Channel: Memory Area Config
  81DCh:0..7   0000 0000 1FF7 R/W DMA Channel: Unknown, usually set to 0300h?
  81DEh:0..7   0000 0000 00FF R/W DMA Channel: Start/Stop/Control
  81E0h:0..7   0000 0000 0000 R   DMA Internal: contains SRC_ADDR_L after DMA
  81E2h:0..7   0000 0000 0000 R   DMA Internal: contains DST_ADDR_L after DMA
  81E4h:0..7   0000 0000 0000 R   DMA Internal: contains SRC_ADDR_H after DMA
  81E6h:0..7   0000 0000 0000 R   DMA Internal: contains DST_ADDR_H after DMA
  81E8h..81FEh 0000 0000 0000 R   Fixed 0000h
```

8200h ICU (interrupts)

```
  8200h        4020 4020 4020 R   ICU Interrupt Pending Flags  (1=Pending)
  8202h        0000 0000 0000 W   ICU Interrupt Acknowledge    (1=Clear)
  8204h        0000 0000 FFFF R/W ICU Interrupt Manual Trigger (1=Set)
  8206h        0000 0000 FFFF R/W ICU Enable Interrupt as int0 (1=Enable)
  8208h        0000 0000 FFFF R/W ICU Enable Interrupt as int1 (1=Enable)
  820Ah        0000 0000 FFFF R/W ICU Enable Interrupt as int2 (1=Enable)
  820Ch        0000 0000 FFFF R/W ICU Enable Interrupt as vint (1=Enable)
  820Eh        2000 0000 FFFF R/W ICU Interrupt Trigger mode (0=Level, 1=Edge)
  8210h        2000 0000 FFFF R/W ICU Interrupt Polarity (0=Normal, 1=Invert)
  8212h+N*4    0003 0000 8003 R/W ICU Vectored Interrupt 0..15 Addr, bit16-31
  8214h+N*4    FC00 0000 FFFF R/W ICU Vectored Interrupt 0..15 Addr, bit0-15
  8252h        0000 0000 FFFF R/W ICU Interrupt Master Disable (1=Off/undoc)
  8254h        0000 0000 5555 R/W Unknown, R/W mask 5555h
  8256h        0000 0000 5555 R/W Unknown, R/W mask 5555h
  8258h..827Eh 0000 0000 0000 R   Mirror of Port 8200h
```

8280h BTDMP Audio (two channels N=0..1, each with speaker out and mic in)

```
  8280h+N*80h  0005 0000 FFFF R/W BTDMP Receive Control         ;\
  8282h+N*80h  0000 0000 7FE7 R/W BTDMP Receive Period          ;
  8284h+N*80h  0000 0000 0FE7 R/W BTDMP Receive Usually 0004h   ;
  8286h+N*80h  0000 0000 0003 R/W BTDMP Receive Usually 0021h   ; RX
  8288h+N*80h  1FFF 0000 1FFF R/W BTDMP Receive Usually 0000h   ; (microphone)
  828Ah+N*80h  0000 0000 0FFF R/W BTDMP Receive Usually 0000h   ;
  828Ch+N*80h  0000 0000 3FFF R/W BTDMP Receive Usually 0000h   ;
  828Eh+N*80h  0000 0000 FFFF R/W BTDMP Receive Usually unused  ;
  8290h+N*80h  0000 0000 FFFF R/W BTDMP Receive Usually unused  ;
  8292h+...    0000 0000 0000 R   Fixed 0000h                   ;
  829Eh+N*80h  0000 0000 8000 R/W BTDMP Receive Enable          ;/
  82A0h+N*80h  0005 0000 FFFF R/W BTDMP Transmit Control        ;\
  82A2h+N*80h  0000 0000 7FE7 R/W BTDMP Transmit Period         ;
```

```
  82A4h+N*80h  0000 0000 0FE7 R/W BTDMP Transmit Usually 0004h  ;
  82A6h+N*80h  0000 0000 0003 R/W BTDMP Transmit Usually 0021h  ; TX
  82A8h+N*80h  1FFF 0000 1FFF R/W BTDMP Transmit Usually 0000h  ; (audio out)
  82AAh+N*80h  0000 0000 0FFF R/W BTDMP Transmit Usually 0000h  ;
  82ACh+N*80h  0000 0000 3FFF R/W BTDMP Transmit Usually 0000h  ;
  82AEh+N*80h  0000 0000 FFFF R/W BTDMP Transmit Usually unused ;
  82B0h+N*80h  0000 0000 FFFF R/W BTDMP Transmit Usually unused ;
  82B2h+...    0000 0000 0000 R   Fixed 0000h                   ;
  82BEh+N*80h  0000 0000 8000 R/W BTDMP Transmit Enable         ;/
  82C0h+N*80h  001x 001F 001F R   BTDMP Receive FIFO Status     ;\
  82C2h+N*80h  0057 005x 0057 R   BTDMP Transmit FIFO Status    ;
  82C4h+N*80h  E0A1 FFFF E0A1 R   BTDMP Receive FIFO Data       ; RX/TX
  82C6h+N*80h  0000 0000 0000 W   BTDMP Transmit FIFO Data      ;
  82C8h+N*80h  0000 0000 0003 R/W BTDMP Receive FIFO Control    ;
  82CAh+N*80h  0000 0000 0003 R/W BTDMP Transmit FIFO Control   ;/
  82CCh+...    0000 0000 0000 R   Fixed 0000h
  8380h..867Eh 03C0 03C0 03C0 R   Mirror of Port 80D6h
```

8680h ? (listed in "xpert_offsets_tbl", but there are just mirrors)

```
  8680h..87FEh 03C0 03C0 03C0 R   Mirror of Port 80D6h
```

8800h ? (listed in "perxteak.per", but seems to be always 0) (dbg aka H2C)

```
  (this is called "Host-to-Core JAM protocol" and consists of 11bit values)
  (not sure if that are MMIO registers, or some 11bit data transfer protocol)
  8800h..8807h 0000 0000 0000 R?  Fixed 0 (reportedly H2C aka dbg stuff?)
```

xpert_offsets_tbl: (with baseIO=8000h)

The DSi's aac.c file contains a "xpert_offsets_tbl" with three sets of I/O regions, which are apparently related to different hardware versions:

```
     ?    JAM  GLUE TMR  SIO  OCEM PMU  APBP AHBM MIU  CRU  DMA  ICU AUDIO ?
  #0 3333 0000 0010 0020 0050 0060 0080 00A0 3333 00C0 3333 0100 0180 0200 3333
  #1 0000 0004 0010 0020 0050 0060 0080 00C0 00E0 0100 0140 0180 0200 0280 0680
  #2 3333 0004 0010 3333 3333 0020 0040 3333 3333 0060 3333 3333 0120 3333 3333
```

All addresses are relative to the MMIO base (usually 8000h), 3333h is apparently some dummy value for unsupported I/O areas.

The DSi does use set #1. The other two sets are probably relicts for older Teak hardware that was never used in DSi consoles (for example, set #2 doesn't event support the APBP region for cmd/reply/semaphore).

DSi Teak MMIO[8000h] - Misc Registers (JAM/GLUE)

MMIO[8004h] - JAM Unknown (R/W)

```
  0-10  Unknown (R/W) (0..7FFh=?)
  11-14 Unused (0)
  15    Unknown (R/W) (0..1=?)
```

Maybe this is for the 11bit "Host-to-Core JAM protocol"?

MMIO[8006h] - JAM Unknown/Crash, DANGER (crashes on read)

Crashes on read.

MMIO[8010h] - GLUE CFG0 (R/W)

GLUE_CFG0 - Number of wait-states for Z-space transactions in predefined zones

Uh, the above name does not match up with below descriptions?

```
  0     Timer 1 clock source (0=107MHz/Core, 1=Timer0_TOUT)
  1     Timer 0 force restart upon Timer 1 output (0=No, 1=Yes)
  2-15  Unused (0)
```

MMIO[8012h] - GLUE Unknown 2bit (R/W)

```
  0-1   Unknown (R/W) (0..3=?)
  2-15  Unused (0)
```

MMIO[8014h] - GLUE Unknown 16bit (R/W)

```
0-15  Unknown (R/W) (0..FFFFh=?)
```

MMIO[8016h] - GLUE Unknown, DSi=0000h or New3DS=00BAh (R)

```
0-15  Unknown (DSi: always 0000h, New3DS: always 00BAh)
```

MMIO[8018h] - GLUE Whatever Parity/Shuffle (R/W)

```
0-3   Value A                                      (R/W)
4     All four bits in Value A XORed together      (R)
5-8   Value B                                      (R/W)
9     All four bits in Value B XORed together      (R)
10-13 Value C                                      (R/W)
14    All four bits in Value C XORed together      (R)
15    Value D                                      (R/W)
```

Unknown purpose, might be a randomizer/maths feature, or an interactive chip-detection feature, or something completely different, like memory-control function, or whatever.

MMIO[801Ah] - GLUE Chip config ID (R)

```
0-15  Fixed, always C902h on DSi and New3DS
```

Used for chip detect (for xpert_offsets_tbl).

More Unknown Registers

XpertTeak was announced to support GPIO, unknown where those registers are located. Probably consist of 16bit direction and 16bit data registers (if it's similar as in older Oak chips). A few GPIO-style bits seem to be in st2/mod0/mod3 CPU registers.

Unknown what below is... maybe MMIO[8800h..8807h] or maybe 11bit values for JAM registers at MMIO[8004h..8006h]... and/or Input-only direction GPIO?

[dbg:800h] - H2C Host-to-Core JAM protocol h2c_0

```
0     Reset                                        (0=No, 1=Yes)
1     Boot                                         (0=No, 1=Yes)
2     Debug                                        (0=No, 1=Yes)
4     URST (user reset)                            (0=No, 1=Yes)
5     -
6     Internal Program (Load code to on-chip memory)  (0=No, 1=Yes)
7-10  -
```

[dbg:801h] - H2C Host-to-Core JAM protocol h2c_1

```
0-6   -
7     Continue core's clock after stopped by software (0=No, 1=Yes)
8     Stop (Stop core's clock)                     (0=No, 1=Yes)
9     NMI                                          (0=No, 1=Yes)
10    Abort                                        (0=No, 1=Yes)
```

[dbg:802h] - H2C Host-to-Core JAM protocol h2c_2

```
0-10  Interrupt 0..10                              (0=No, 1=Yes)
```

[dbg:803h] - H2C Host-to-Core JAM protocol h2c_3

```
0-10  GPI (General Purpose Input) 0..10            (0=Low, 1=High)
```

[dbg:804h] - H2C Host-to-Core JAM protocol h2c_4

```
0-4   GPI (General Purpose Input) 11..15           (0=Low, 1=High)
5-8   -
9-10  UI (User Input) 0..1                         (0=Low, 1=High)
```

[dbg:805h] - H2C Host-to-Core JAM protocol h2c_5

```
0-10  Interrupt external/internal control 0..10    (0=Ext, 1=Int)
```

[dbg:806h] - H2C Host-to-Core JAM protocol h2c_6

```
  0-10  GPI Enable Control 0..10                        (0=Disable, 1=Enable)
```

[dbg:807h] - H2C Host-to-Core JAM protocol h2c_7

```
  0-4   GPI Enable Control 11..15                       (0=Disable, 1=Enable)
  5-8
  9-10  User Input Enable Control 0..1                  (0=Disable, 1=Enable)
```

DSi Teak MMIO[8020h] - Timers (TMR)

MMIO[8020h/8030h] - Timer 0/1 Control (R/W)

```
  0-1   Time prescaler (0=Div1, 1=Div2, 2=Div4, 3=Div16)
  2-4   Count mode
         0h: single count     Stop at zero
         1h: auto restart     Wrap from zero to Reload value
         2h: free running     Wrap from zero to FFFFFFFFh
         3h: event count      Decrement manually, and stop at zero
         4h: watchdog mode 1  Trigger Teak Reset at zero
         5h: watchdog mode 2  Trigger Teak NMI at zero
         6h: watchdog mode 3  Trigger Unacknowledgeable-Timer-IRQ at zero?
         7h: reserved         Same as mode 0 (stop at zero)
  5     Unused (0)
  6     Output signal polarity                (0=Normal, 1=Invert/Buggy?)
  7     Clear output signal; when bit14-15=0  (0=No change, 1=Clear)   (W)
  8     Pause the counter                     (0=Unpause, 1=Pause)
  9     Freeze COUNTER_L/H register value     (0=Freeze, 1=Update)
         Note: Bit8/Bit9 can be forced to always 1 via other Timer's Bit13
  10    Restart/Reload the counter            (0=No change, 1=Restart) (W)
  11    Breakpoint requests enable   (0=Disable, 1=Enable) (DANGER/TRAP)
  12    Clock source (0=InternalClk=134MHz/107MHz, 1=ExternalClk=None)
  13    General Purpose (somehow interact between Timer 0 and 1)
         Timer0: Force Timer1.Control.Bit9=1  (0=No, 1=Yes/ForceUpdate)
         Timer1: Force Timer0.Control.Bit8=1  (0=No, 1=Yes/ForcePause)
  14-15 Clear output signal automatically (0=No, 1/2/3=After 2/4/8 cycles)
```

The InternalClk is different for Timer 0 and 1, and depends on GLUE setting:

```
  For Timer 0 (no "waitstates")   --> 134.055928MHz
  For Timer 1 (with "waitstates") --> 107.244742MHz (aka 134.055928MHz/1.25)
  For Timer 1 (if GLUE_CFG0.bit0) --> Timer0_TOUT (bugs if Timer0_Reload<3)
  Unknown what "waitstates" refers to, probably not the Z0/Z1/Z2/Z3 waits?
```

Bit10 is automatically cleared after writing - however, when using prescaler other than Div1, it may take a few cycles until bit10 gets cleared (and perhaps also takes time until it is processed??? if so, one shouldn't clear bit10 in that timeslot).

Bit7 is usually cleared after writing - but not always, still unknown what/when that happens?

MMIO[8022h/8032h] - Timer 0/1 Trigger Event/Watchdog (W)

```
  0     In Event Mode: Decrement Counter (0=No change, 1=Decrement)
        In Watchdog Mode: Reload Counter (0=No change, 1=Reload)
  1-15  Unused (0)
```

MMIO[8024h/8034h] - Timer 0/1 Reload value, bit0-15 (R/W)
MMIO[8026h/8036h] - Timer 0/1 Reload value, bit16-31 (R/W)

```
  0-31  Start/Reload value for decrementing counter
```

Used as start/reload value; copied to counter when setting Control.bit10, or after reaching 0 in auto-restart mode, or when writting trigger bit in watchdog mode.

Not used in free-run mode (which starts at current counter value, and wraps from 0 to FFFFFFFFh).

MMIO[8028h/8038h] - Timer 0/1 Counter value, bit0-15 (R)
MMIO[802Ah/803Ah] - Timer 0/1 Counter value, bit16-31 (R)

```
  0-31  Current (or frozen) decrementing counter value
```

When using freeze, counter keeps decreasing in background, but reading the counter register returns the frozen value (latched when changing Control.bit9 from 1-to-0).

MMIO[802Ch/803Ch] - Timer 0/1 PWM Reload value, bit0-15 (R/W)
MMIO[802Eh/803Eh] - Timer 0/1 PWM Reload value, bit16-31 (R/W)

```
  0-31  Restart value for PWM counter (uh, maybe PWM duty?)
```

Unknown if/when/how this is used (maybe just goes to an unimplemented PWM output pin).

DSi Teak MMIO[8050h] - Serial Port (SIO)

This is a SPI-style serial I/O interface, allowing to connect some peripheral or debugging hardware (unused in DSi/3DS).

MMIO[8050h] - SIO Control (R/W)

```
  0     Chip Select Polarity       (0=Active High, 1=Active Low)
  1     Chip Select Output         (0=Disable/Hangs, 1=Enable) (for Master)
  2     Master/Slave Clock         (0=FromClkDivider, 1=ExternalClk/Hangs)
  3     Clock Polarity             (0=Idle Low, 1=Idle High)
  4     Clock Edge Phase           (0=InputOnRising, 1=OutputOnRising)
  5     Transfer End Interrupt     (0=Enable, 1=Disable/Hangs/NoStatusDone)
  6-11  Unused (0)
  12-15 Num data bits per transfer (0=Hangs, 1..15=2bit..16bit)
```

MMIO[8052h] - SIO Clock Divider (R/W)

```
  0-6   Clock Divider 1            (1..7Fh = Div1..Div127) (0=Div1, too)
  7     Unused (0)
  8-14  Clock Divider 2            (1..7Fh = Div1..Div127) (0=Div1, too)
  15    Unused (0)
```

The transfer shift clock is 134MHz/(Divider1*Divider2).

MMIO[8054h] - SIO Data (R) and (W)

```
  0-15  Transfer data (probably using only LSBs when NumBits<16)
```

Writing the transmit data starts the transfer, the reply data can be read when Transfer Done status flag gets set.

MMIO[8056h] - SIO Enable (R/W)

```
  0     Enable SIO operation       (0=Disable/Hangs, 1=Enable)
  1-15  Unused (0)
```

MMIO[8058h] - SIO Status (R)

```
  0     Transfer done   (0=No, 1=Done, Data can be read now)
  1     Overrun error   (0=No, 1=New data arrived before reading old data)
  2-15  Unused (0)
```

Bit0,1 are automatically cleared after reading the status register.

SIO Transfer Timing Bugs
After transfer end, one must apparently wait at least "Divider1*Divider2/2" cycles before starting another transfer, else the new transfer won't start.
There are two dummy transfer clocks inserted (so transferring 2..16 bits does actually take "(4..18)*Divider1*Divider2" cycles).
And, transfer start doesn't seem to take place until next clock boundary (that may take between "0 and 0.999*Divider1*Divider2" extra cycles).

DSi Teak MMIO[8060h] - Debug (OCEM, On-chip Emulation Module)

MMIO[8060h] - OCEM Program Flow Trace Buffer (PFT), bit0-15 (R)
MMIO[8061h] - OCEM Program Flow Trace Buffer (PFT), bit16-31 (R)
This buffer contains addresses of the 16 most recent jump opcodes, reading returns the oldest of those 16 addresses (or 16 random address/page values if read after reset). One can disable buffer updating via MMIO[8070h] (eg. to avoid jumps inside of a debug handler to be written to the buffer). There is "buffer full" condition that gets set 16 jumps after reset, and that can trap an optional buffer full exception.

```
0-17   Program Flow Trace Address
18-23  Unused (0)
24-27  Program Flow Trace Page
28-31  Unused (0)
```

Unknown if there is any way to "make the buffer empty", either by clearing the buffer, or by removing entries until it gets empty. Also, there is no known way to read the next buffer entry, except for one trick: temporarily re-enable buffer updates via MMIO[8070h] and issue a dummy jump; this will add a new entry, and reading will then return the next oldest entry; by that, the buffer won't get empty (but, when ignoring the address of the dummy jump, one can extract the actual non-dummy data from the buffer).

MMIO[8062h/8064h/8066h] - OCEM Program Break Address 1/2/3, bit0-15 (R/W)
MMIO[8063h/8065h/8067h] - OCEM Program Break Address 1/2/3, bit16-31 (R/W)

```
0-17   Program Break Address
18-23  Unused (0)
24-27  Program Break Page (should be usually 00h)
28-31  Unused (0)
```

MMIO[8068h/8069h/806Ah] - OCEM Program Break Counter 1/2/3 (R/W)

```
0-7    Program Address Break Counter (decrements upon PC=break.addr)
8-15   Unused (0)
```

Break triggers upon count zero. Writing count=NNh will trap after program counter had hit the break address NNh times (writing count=00h will trigger immediately, without even hitting the break address).

MMIO[806Bh] - OCEM Data Break Mask (R/W)
MMIO[806Ch] - OCEM Data Break Address (R/W)

```
0-15   Mask/Address
```

MMIO[806Dh] - OCEM Breakpoint Enable Flags DANGER (0=Disable, 1=Enable/Trap)

```
0      Data value break point on data write transaction
1      Data value break point on data read transaction
2      Data address break point as a result on data write transaction
3      Data address break point as a result on data read transaction
4      Simultaneous data address and data value match
5      External register write transaction  ;\(aka ext0/1/2/3?)
6      External register read transaction   ;/
7      Program Address break 1 count zero
8      Program Address break 2 count zero
9      Program Address break 3 count zero
10     Break on any program jumps instead of executing the next address
11     Break on detection of interrupt service routine
12     Break as a result of program flow trace buffer full
13     Break when returning to the beginning of block repeat loop
14     Break on illegal condition (uh, are that... illegal opcodes?)
15     Single Step
```

Note: Data value refers to the "dvm" CPU-core register. Data write matches can occur on things like "mov r0,1234h" or "addv r0,1" (when r0 becomes equal to dvm).

MMIO[806Eh] - OCEM Mode/Indication? (R/W)

```
0      Program Flow Trace Buffer full (0=Not full/OldestIsGarbage, 1=Full)
1-11   Unknown (R/W)
```

```
  12     MOVD instruction detected (uh, usually 1, even when not using movd?)
  13     User reset activated while in break point service routine
  14     Boot mode   (0=No, 1=Yes)
  15     Debug mode  (0=No, 1=Yes)
```
Unknown if this is a mode or status register, or mixup thereof?

MMIO[806Fh] - OCEM Breakpoint Status Flags (R/W)
```
  0      Break caused by Data value match
  1      Break caused by Data address match
  2      Break caused by Data value and data address match
  3      Break caused by User defined register transaction (aka ext0/1/2/3?)
  4      Break caused by an external event (aka what?)
  5      Break caused by Program address break 1 count zero
  6      Break caused by Program address break 2 count zero
  7      Break caused by Program address break 3 count zero
  8-10   Unknown (R/W)
  11     Break caused by Branch break point
  12     Break caused by Interrupt break point
  13     Break caused by Program Flow Trace Buffer full
  14     Break caused by Illegal break point
  15     Break caused by Software trap
```
Uh, R/W mask is BFFFh, ie. bit14 is always 0, that doesn't match up with above description?

MMIO[8070h] - OCEM Program Flow Trace Update Disable (R/W)
```
  0      Disable Program Flow Buffer Updating (0=Enable, 1=Disable)
  1-15   Unused (0)
```

MMIO[8072h] - Unknown 16bit? (R/W)
```
  0-15   Unknown (R/W)
```
Maybe be "programming model signature" mentioned in perxteak.per (though that is claimed to use only upper 8bit)? That signature might be some general purpose value to notify the debugger what kind of code is executed?

MMIO[8074h] - OCEM Boot/Debug Mode (R)
```
  0-13   Unused? (0)
  14     Boot mode   (0=No, 1=Yes)
  15     Debug mode  (0=No, 1=Yes)
```
Usually contains the same mode bits as written to MMIO[806Eh].bit14/bit15 (maybe one of the registers contains the OLD mode bits; when entering a debug handler or so).

DSi Teak MMIO[8080h] - PLL and Power (PMU, Power Management Unit)

MMIO[8080h] - PMU PLL Multiplier (R/W)
```
  0-15  Configuration of the PLL clock multiplication (0..FFFFh=what?)
```
Power-up: C00Eh. Unknown what that does... multiplying clock by 16bit seems unlikely. Maybe alike CCR2 register in files peroak.per and pertklc.per?
Or maybe 0Ch and 0Eh translate to (14+1):12 ratio (ie. the 134MHz/1.25 divider; though changing this register doesn't seem to affect that)?

MMIO[8082h] - PMU PLL Power-on config (R/W)
```
  0     PLL power-on configuration value for PLL use (0..1=what)
  1-15  Unused (0)
```
Power-up: 0001h. Unknown what that is. Default value for PLB bit in next register? Or maybe something needed for powering-up the PLL (when clearing bypass flag)?

MMIO[8084h] - PMU PLL Divider/Bypass (R/W?)
```
  0-6   Clock Divider (0 or 1=Div1) (2..7Fh=Crashes?)
```

```
  7-14  Unused (0)
  15    Bypass PLL    (0=Use PLL/Crashes, 1=Bypass; works only if Div1)
```
Power-up: 8000h. Ie. PLL bypassed, and using 134MHz clock from DSi as is.
Unknown how to change/enable the PLL without crashing.

MMIO[8086h,809Ah] - PMU Wake/Shutdown module(s) (0=Wake-up, 1=Shutdown) (R/W)
MMIO[8088h,809Ch] - PMU Recover module(s) on interrupt 0 (R/W)
MMIO[808Ah,809Eh] - PMU Recover module(s) on interrupt 1 (R/W)
MMIO[808Ch,80A0h] - PMU Recover module(s) on interrupt 2 (R/W)
MMIO[808Eh,80A2h] - PMU Recover module(s) on vectored interrupt (R/W)
MMIO[8090h,80A4h] - PMU Recover module(s) on Timer 0 (except bit8) (R/W)
MMIO[8092h,80A6h] - PMU Recover module(s) on Timer 1 (except bit9) (R/W)
MMIO[8094h,80A8h] - PMU Recover module(s) on non-maskable interrupt (R/W)
MMIO[8096h] - PMU Recover DMA module on external signal (bit1 only) (R/W)
```
 For registers MMIO[8086h..8096h]:
  0     Core     ;aka cpu?
  1     DMA
  2     SIO
  3     GLUE
  4     APBP/HPI
  5     AHBM
  6     Unused (0)
  7     OCEM
  8     Timer 0
  9     Timer 1
  10    JAM
  11-15 Unused (0)
 For registers MMIO[809Ah..80A8h]:
  0     BTDMP 0
  1     BTDMP 1
  2-15  Unused (0)
```
Register 8090h.bit8 and 8092h.bit9 are unused, always 0 (ie. a disabled timer cannot wakeup itself).
Register 8096h supports bit1 only (DMA), all other bits are unused, always 0.

MMIO[8098h] - BM: PMU Breakpoint mask module(s) (bit1,8,9 only) (R/W)
```
  0     Unused (0)
  1     DMA
  2-7   Unused (0)
  8     Timer 0
  9     Timer 1
  10-15 Unused (0)
```
Unknown what this does. It looks similar to the "Recover" registers, but, going by the description, it is "breakpoink masking" something instead of "recovering".

MMIO[80AAh] - 16bit R/W ?
MMIO[80ACh] - 16bit R/W ?
MMIO[80AEh] - 16bit R/W ?

DSi Teak MMIO[80C0h] - Host Port Interface (APBP aka HPI)

The APBP (aka HPI Host Port Interface) registers are used to communicate between ARM and DSP.

MMIO[80C0h/80C4h/80C8h] - APBP DSP-to-ARM Reply 0/1/2 (R/W)
MMIO[80C2h/80C6h/80CAh] - APBP ARM-to-DSP Command 0/1/2 (R)
```
  0-15   Command/Reply Data
```

MMIO[80CCh] - APBP DSP-to-ARM Semaphore Set Flags (0=Clear, 1=Set) (R/W)

MMIO[80CEh] - APBP ARM-to-DSP Semaphore Interrupt Mask (?=On/Off) (R/W)
MMIO[80D0h] - APBP ARM-to-DSP Semaphore Ack Flags (0=No change, 1=Clear) (W?)
MMIO[80D2h] - APBP ARM-to-DSP Semaphore Get Flags (R)

```
0-15   Semaphore Flag 0..15
```

MMIO[80D4h] - APBP Control (R/W)

```
0-1    Unused (0)
2      ARM-side register endianness (0=Normal, 1=Big-Endian/DANGER)
3-7    Unused (0)
8      Interrupt when CMD0 is written by ARM (0=Enable, 1=Disable)
9-11   Unused (0)
12     Interrupt when CMD1 is written by ARM (0=Enable, 1=Disable)
13     Interrupt when CMD2 is written by ARM (0=Enable, 1=Disable)
14-15  Unused (0)
```

MMIO[80D6h] - APBP DSP-side Status (R)

```
0-4    Unused? (usually 0)
5      Reply Register 0 Read Flag     (0=Was Read by ARM?, 1=No)
6      Reply Register 1 Read Flag     (0=Was Read by ARM?, 1=No)
7      Reply Register 2 Read Flag     (0=Was Read by ARM?, 1=No)
8      Command Register 0 Update Flag (0=Was Written by ARM?, 1=No)
9      Semaphore IRQ Flag             (0=No, 1=[80D2h] AND NOT [80CEh])
10-11  Unused? (usually 0)
12     Command Register 1 Update Flag (0=Was Written by ARM?, 1=No)
13     Command Register 2 Update Flag (0=Was Written by ARM?, 1=No)
14-15  Unused? (usually 0)
```

MMIO[80D8h] - APBP ARM-side Status (mirror of ARM9 Port 400430Ch) (R)

```
0      Read Transfer Underway Flag    (0=No, 1=Yes/From DSP Memory)
1      Write Transfer Underway Flag   (0=No, 1=Yes/To DSP Memory)
2      Peripheral Reset Flag          (0=No/Ready, 1=Reset/Busy)
3-4    Unused (0)
5      Read FIFO Full Flag            (0=No, 1=Yes/Full)
6      Read FIFO Not-Empty Flag       (0=No, 1=Yes, ARM9 may read PDATA)
7      Write FIFO Full Flag           (0=No, 1=Yes/Full)
8      Write FIFO Empty Flag          (0=No, 1=Yes/Empty)
9      Semaphore IRQ Flag             (0=None, 1=IRQ)
10     Reply Register 0 Update Flag   (0=Was Written by DSP, 1=No)
11     Reply Register 1 Update Flag   (0=Was Written by DSP, 1=No)
12     Reply Register 2 Update Flag   (0=Was Written by DSP, 1=No)
13     Command Register 0 Read Flag   (0=Was Read by DSP, 1=No)
14     Command Register 1 Read Flag   (0=Was Read by DSP, 1=No)
15     Command Register 2 Read Flag   (0=Was Read by DSP, 1=No)
```

DSi Teak MMIO[80E0h] - AHBM - Advanced High Performance Bus Master

AHBM does allow the Teak CPU to access the ARM-side memory (in combination with DMA interface at MMIO[8180h]).

Supported ARM-side memory areas
DSi allows to access most of the ARM9 address space:

```
02000000h/Main RAM     --> works
03000000h/Shared RAM   --> works (maybe also New Shared RAM, if any mapped?)
04000000h/ARM9 I/O     --> works
05000000h/Palette      --> works
06000000h/VRAM         --> works
07000000h/OAM          --> works
08000000h/GBA SLOT ROM --> works (with dummy FFFFFFFFh values)
```

```
  0A000000h/GBA SLOT RAM --> works (with dummy FFFFFFFFh values)
  FFFF0000h/ARM9 BIOS    --> works (lower 32K only, upper 32K is zerofilled)
  DTCM/ITCM              --> probably ignored?
  Any other              --> returns zero (but without MMIO[80E0h] error flag)
```
3DS in 3DS mode is more restrictive:
```
  20000000h/FCRAM        --> works
  1FF80000h/AXI          --> works
  1FF40000h/DSP/DATA     --> oddly mirrors to 1FF80000h/AXI
  Any other              --> rejected (and sets MMIO[80E0h].bit4 error flag)
```

MMIO[80E0h] - AHBM Status (R)
```
  0-1   Usually/always 0 ?
  2     Burst queue not empty  (0=Empty, 1=Not-empty)
  3     Usually/always 0 ?
  4     Busy/stuck/error?  (0=Normal, 1=Invalid ARM address)  ;3DS only?
  5-15  Usually/always 0 ?
```
"Applications wait for all bits to be 0 before connecting AHBM to DMA."

MMIO[80E2h+(0..2)*6] - AHBM Channel 0..2 Configure Burst/Data (R/W)
```
  0     Applications set this to 1 if BURST is non-zero, uh?
  1-2   Burst type     (0=x1, 1=x4, 2=x8, 3=?)
  3     Unknown (R/W)  (0=Normal, 1=Dunno/NoTransfer?)
  4-5   Data type      (0=8bit, 1=16bit, 2=32bit, 3=?)
  6     Unused (0)
  7     Unknown (R/W)  (0=Normal, 1=Dunno/TransferHangs/crashes?)
  8-11  Unknown (R/W)  (0..Fh=?) (usually 0)
  12-15 Unused (0)
```
Used values: 0010h=?,0020h=?,0025h=dma?

MMIO[80E4h+(0..2)*6] - AHBM Channel 0..2 Configure Whatever (R/W)
```
  0-7   Unknown (R/W)      (0..FFh=?) (usually 0)
  8     Transfer direction (0=Read external memory, 1=Write external memory)
  9     Applications always set this (usually 1=set) (but also works when 0)
  10-15 Unused (0)
```
Used values: 0200h=read, 0300h=write

MMIO[80E6h+(0..2)*6] - AHBM Channel 0..2 Configure DMA (R/W)
```
  0-7   Connect to DMA channel 0..7   (0=No, 1=Connect)
  8-15  Unused (0)
```
Used values: bit0-7=dma0..7, 0000h=All off.

MMIO[80F4h] - Unknown 6bit? bit10-15 are used (R/W)
MMIO[80F6h] - AHBM Internal FIFO (R) and maybe also (W?)
MMIO[80F8h] - Unknown always zero? (R?)
MMIO[80FAh] - Read/write-able(!) mirror of MMIO[80FCh] (R/W)
MMIO[80FCh] - Unknown 16bit? (R/W)
MMIO[80FEh] - Unknown 16bit? (R/W)
Unknown registers, normally left unused. The FIFO register seems to contain the most recent 8x16bit values that were transferred via AHBM. Unknown if that FIFO (and whatever address setting) could be used for manual (non-DMA) AHBM transfers.

DSi Teak MMIO[8100h] - Memory Interface Unit (MIU)

MMIO[8100h] - MIU Waitstate Settings, bit0-15 (R/W)
MMIO[8102h] - MIU Waitstate Settings, bit16-31 (R/W)
```
  0-3    Number of wait-states for off-chip Z0 block
  4-7    Number of wait-states for off-chip Z1 block
```

```
  8-11   Number of wait-states for off-chip Z2 block
  12-15  Number of wait-states for off-chip Z3 block
  16-19  Number of wait-states for off-chip Z blocks outside Z0/Z1/Z2/Z3
  20-23  Number of wait-states for off-chip X/Y memory transactions
  24-27  Number of wait-states for off-chip Program-memory transactions
  28-31  Unused (0)
```
Uh, what means "off-chip" exactly, what kind of memory, on what "chip"?

MMIO[8104h] - MIU Waitstate Area Z0 (R/W)
MMIO[8106h] - MIU Waitstate Area Z1 (R/W)
MMIO[8108h] - MIU Waitstate Area Z2 (R/W)
MMIO[810Ah] - MIU Waitstate Area Z3 (R/W)
```
  0-5    Wait-state block Start address in 1K-word units
  6-11   Wait-state block End address in 1K-word units
  12-15  Wait-state block Page (lower 4bit of page)
```

MMIO[810Eh] - MIU X Page (or unused) (16bit) (R/W)
MMIO[8110h] - MIU Y Page (or unused) (8bit) (R/W)
MMIO[8112h] - MIU Z Page (or "absolute" page) (16bit) (R/W)
The X/Z pages are 16bit wide:
```
  0-15   Memory Page (...base or so, in WHAT-units?)
```
The Y page is only 8bit wide:
```
  0-7    Memory Page (...base or so, in WHAT-units?)
  8-15   Unused (0)
```
The register meaning depends on the PGM paging mode:
```
  When PGM=0 --> MMIO[8112h] is the "absolute" data memory page
  When PGM=1 --> MMIO[810Eh/8110h/8112h] are the X/Y/Z pages
```

MMIO[8114h] - MIU X/Y Page Size for Page 0 (or all pages) (R/W)
MMIO[8116h] - MIU X/Y Page Size for Page 1 (or unused) (R/W)
MMIO[8118h] - MIU X/Y Page Size for Off-chip Pages (or unused) (R/W)
```
  0-5    X memory size (0..3Fh)
  6-7    Unused (0)
  8-14   Y memory size (1..40h)
  15     Unused (0)
```
The register meaning depends on the PGM paging mode:
```
  When PGM=0?--> MMIO[8114h,8116h,8118h] used for Page 0, 1, and Off-chip pages
  When PGM=1?--> MMIO[8114h] used for all pages
```
For Y memory size, the hardware does automatically replace the written value by min=01h or max=40h when trying to to write 00h or 41h-7Fh.

MMIO[811Ah] - MIU Config for Misc stuff (R/W)
```
  0      Program protection mechanism, uh, what is that? (0=Disable, 1=Enable)
  1      Program page for entire program space (from Test pin) (1=offchip)
  2      Program page for breakpoint handler (0=Page1/offchip, 1=Page0/onchip)
  3      Unknown (R/W)  (0..1=?)
  4      Core/DMA data use only Z-even address bus (single? PGM=0?) (1=Enable)
  5      Unknown (R/W)  (0..1=?)
  6      Paging mode (PGM) (for X/Y/Z pages)  (0=Normal, 1=DANGER)
  7-15   Unused (0)
```
Power-on default is 0014h. Bit1-2 are read-only (always bit1=0, bit2=1).

MMIO[811Ch] - MIU Config for Program Page and Download Memory (R/W)
```
  0      Select Z-space data memory (0=Regular Memory, 1=Download memory)
  1      Download mem is/was selected (bit1 can be cleared during Trap only)
  2-5    Alternative Program Page for movd/movp opcodes (PDLPAGE) (4bit)
  6      Select program page for movd/movp (0=2bit/movpd, 1=4bit/PDLPAGE)
  7-15   Unused (0)
```

Bit1 gets set when setting bit0=1 (bit1 is usually readonly, except, trap handler may clear bit1 by writing to it).
Unknown what Download means, maybe off-chip data memory?

MMIO[811Eh] - MIU Base Address for MMIO Registers (R/W)

```
0-9    Unused (0)
10-15  MMIO Base Address (in 400h-word units)
```

Power-on default is 8000h, ie. mapping MMIO[8000h..87FFh] to 8000h..87FFh.

MMIO[8120h] - MIU Observability Mode (R/W)

```
0      Observability Enable (0=Disable, 1=Enable)
1-3    Observability Mode   (0..4=Mode, see below)
4-15   Unused (0)
```

Observability modes (defines which Core/DMA address/data buses are reflected on the off-chip XZ buses):

```
00h: Core XZ address/data buses
01h: Core Y address/data buses
02h: Core P address/data buses
03h: DMA DST address/data buses
04h: DMA SRC address/data buses
```

MMIO[8122h] - MIU Pin Config? (R/W)

```
0     Z Read Polarity Bit   - for DRZON/DRZEN
1     Z Write Polarity Bit  - for DWZON/DWZEN
2     Z Strobe Polarity Bit - for ZSTRB
3     X Strobe Polarity Bit - for XSTRB
4     X Select Polarity Bit - for XS
5     Z Select Polarity Bit - for ZS
6     Signal Polarity Bit   - for RD_WR
7-15  Unused (0)
```

DSi Teak MMIO[8140h] - Code Replacement Unit (CRU)

The CRU is intended to fix bugs in ROM-based programs (rather useless for RAM-based DSi code).

MMIO[8140h+(0..15)*4] - CRU Entry 0..15 Offset, bit0-15 (R/W)
MMIO[8142h+(0..15)*4] - CRU Entry 0..15 Offset, bit16-31 (R/W)

```
0-17   Program Address                          (0..3FFFFh)           (R/W)
18-21  Program Page (usually 0)                 (0..Fh)               (R/W)
22-30  Unused (0) (except bit25-30 in entry 15)
25-28  Entry 15: Match entry number             (0..15=Entry 0..15)   (R)
29     Entry 15: Match flag (cleared after read) (0=None, 1=Yes/match) (R)
30     Entry 15: Master enable for all entries   (0=Disable, 1=Enable) (R/W)
31     Enable Entry                             (0=Disable, 1=Enable) (R/W)
```

There are sixteen identical entries, resembling OCEM program breaks, with some extra control/status bits in last entry.
The feature is working in so far that the read-only status bits get set upon matches, but the match should reportedly insert "a branch instruction" or trigger a "trap" exception, which doesn't happen.
Unknown if there are further config/mode/enable bits elsewhere for activating that behaviour. To bypass ROM code, it might redirect to data RAM, or external AHBM memory?

DSi Teak MMIO[8180h] - Direct Memory Access (DMA)

DMA Control/Status

MMIO[8184h] - DMA Channel Start Flags (0=Stop/Done, 1=Start/Busy) (R/W)
MMIO[8186h] - DMA Channel Pause Flags (0=Normal, 1=Pause) (R/W)

```
  0-7    Channel 0..7 flags
  8-15   Unused (0)
```
Transfer start/stop should be done via [81DEh].bit14/15 (using RMW to change [8184h] directly could mess up other channels; if they finish during the RMW).
Channel 0 is automatically enabled on power up, because it is used for ARM-side data transfers via DSP_PDATA (Port 4004300h).

MMIO[8188h] - DMA Channel End Flags for Size0 (R)
MMIO[818Ah] - DMA Channel End Flags for Size1 (R)
MMIO[818Ch] - DMA Channel End Flags for Size2 (R) (all done)
```
  0-7    Channel 0..7 end flags (0=No, 1=End)
  8-15   Unused (0)
```
The COFF labels refer to these three registers as "seox", whatever that means.
The 3rd register can be used to check for transfer completion (transfer end does also clear the Start/Enable bit in MMIO[8184h]).

MMIO[818Eh] - DMA Whatever Slot Config, bit0-15 (R/W)
MMIO[8190h] - DMA Whatever Slot Config, bit16-31 (R/W)
```
  0-2    Whatever (0..7) (initially 0 on reset)
  4-6    Whatever (0..7) (initially 1 on reset)
  8-10   Whatever (0..7) (initially 2 on reset)
  12-14  Whatever (0..7) (initially 3 on reset)
  16-18  Whatever (0..7) (initially 4 on reset)
  20-22  Whatever (0..7) (initially 5 on reset)
  24-24  Whatever (0..7) (initially 6 on reset)
  28-30  Whatever (0..7) (initially 7 on reset)
  Bit3,7,11,15,19,23,25,31 are unused (always 0)
```
Reportedly "some sort of slots for channel 0..7" maybe priority for DMA 0..7?
transfer can hang (eg. when changing 1st word to 4444h or higher)?

_____ DMA Channels _____

MMIO[81BEh] - DMA Select Channel (R/W)
```
  0-2    Select the channel to be mapped to MMIO[81C0h..81Exh] (0..7)
  3-15   Unused (0)
```

MMIO[81C0h:0..7] - DMA Channel: Source Address, bit0-15 (R/W)
MMIO[81C2h:0..7] - DMA Channel: Source Address, bit16-31 (R/W)
MMIO[81C4h:0..7] - DMA Channel: Destination Address, bit0-15 (R/W)
MMIO[81C6h:0..7] - DMA Channel: Destination Address, bit16-31 (R/W)
```
  0-31   Address (within the selected memory area, see MMIO[81DAh])
```
Unknown how/what works exactly...
DSP/data probably uses only lower 18bit & probably ignores X/Y/Z and MMIOBASE
DSP/code probably uses only lower 18bit & probably ignores prpage/movpd
DSP/mmio probably uses only lower 11bit & probably ignores MMIOBASE
ARM/ahbm uses full 32bit address, counted in BYTE-units (not 16bit-units)

MMIO[81C8h:0..7] - DMA Channel: Size0 (inner dimension) (R/W)
MMIO[81CAh:0..7] - DMA Channel: Size1 (middle dimension) (R/W)
MMIO[81CCh:0..7] - DMA Channel: Size2 (outer dimension) (R/W)
```
  0-15   Length (for each array dimension)   (0001h..FFFFh)
```
Trying to write size=0000h is automatically replaced by size=0001h.
Total transfer length is Size0*Size1*Size2 (that is always counted in 16bit units; Size0 should contain an even number when using 32bit mode).
Normal 1-dimensional transfer would use Size0=Len, Size1=Size2=1. Size1 can be useful for 2-dimensional arrays. Size2 was announced to be useful for re-ordering things like RGB values (uh, but that may require 8bit-addressing, and negative step values, or separate step-direction flags?).

MMIO[81CEh:0..7] - DMA Channel: Source Step0 (R/W)
MMIO[81D0h:0..7] - DMA Channel: Source Step1 (R/W)
MMIO[81D2h:0..7] - DMA Channel: Source Step2 (R/W)
MMIO[81D4h:0..7] - DMA Channel: Destination Step0 (R/W)
MMIO[81D6h:0..7] - DMA Channel: Destination Step1 (R/W)
MMIO[81D8h:0..7] - DMA Channel: Destination Step2 (R/W)

```
0-15   Step (... in 8bit/16bit/32bit units?) (signed or unsigned?)
```

MMIO[81DAh:0..7] - DMA Channel: Memory Area Config (R/W)

```
0-3    Source Memory Area      (0..0Fh, see below)
4-7    Destination Memory Area (0..0Fh, see below)
8      Unknown?                (0=Normal, 1=No Irq, No end, maybe repeat?)
9      Different Memory Areas  (0=No/Slow, 1=Yes/Simultaneous Read+Write)
10     Transfer Unit size      (0=16bit/Slow, 1=32bit/Fast)
11     Unused (0)
12-13  Unknown?                (0..3=?)
12-15  Transfer Speed          (0=Slow, 1/2=Medium, 3=Fast) (or burst size?)
```

Source/Destination Memory Areas can be:

```
00h DSP/Data memory                                ;\
01h DSP/MMIO registers                             ; 16bit-address units
05h DSP/Code memory (only for DST_SPACE) (untested) ;/
07h ARM/AHBM external memory (via AHBM registers)   ;-8bit-address units
```

Used values: 670h,607h,400h,250h. Reset value is F200h (and kept so for channel 0).

MMIO[81DCh:0..7] - DMA Channel: Unknown, usually set to 0300h? (R/W)

```
0-2    Unknown (R/W)  (0..07h=?) (usually 0)
3      Unused (0)
4-7    Unknown (R/W)  (0..0Fh=?) (usually 0h)
8-9    Unknown (R/W)  (0/1/2=Hangs?, 3=Normal)
10-12  Unknown (R/W)  (0..07h=?) (usually 0h)
13-15  Unused (0)
```

Usually set to 0300h or 0700h (or, set to 0011h for DSP_PDATA transfer for channel 0, though that does also work with reset value 0000h).

MMIO[81DEh:0..7] - DMA Channel: Start/Stop/Control (R/W)

```
0-2   Interrupt upon Size 0..2 End (0=Disable, 1=Enable)               (R/W)
3-5   Never set Size2 End flag? (0=Normal, 1=No end, maybe repeat?)    (R/W)
6-7   Unknown (0..3)                                                  (R/W)
8-13  Unused? (0)
14-15 Start/Stop Transfer (0=No change, 1=Start, 2=Stop, 3=Same as 1)    (W)
```

DMA Unknown/Internal Registers

MMIO[8192h] - Unknown, one register with RW-mask 7C03h (R/W)
Unknown/internal stuff. Writing 0000h, 0001h, 0002h, 0800h, 1000h, 4000h, FFFFh passes okay, however, writing 0400h or 2000h causes TRAP exception. Note: Don't remember how that had happened; it might have been caused indirectly by dvm data matches.

MMIO[81B6h..81BCh] - Unknown, four registers with RW-mask FFFFh (R/W)
Unknown/internal stuff. The four 16bit R/W registers might be general purpose fill values (if they have no other purpose)?

MMIO[8180h] - DMA Internal: Channel Size0 Busy or so? (R)
MMIO[8182h] - DMA Internal: Channel Size1 Busy or so? (R)
Unknown, can contain similar busy flags as in [8184h], although in read-only form, and not exactly the same value (eg. bit0 tends to be zero despite of channel being enabled for DSP_PATA transfers). Maybe related to Size0/1, or maybe indicating which channel is RIGHT NOW transferring data (if so, only one bit should be seen

set at a time; or maybe bits go off one after another during processing of multiple channels).

MMIO[8194h] - DMA Internal: initially 0, contains SRC_ADDR_L after DMA (R)
MMIO[8196h] - DMA Internal: initially 0, contains DST_ADDR_L after DMA (R)
MMIO[8198h..81B4h] - Unknown, fixed 0 (R)
MMIO[81E0h:0..7] - DMA Internal: initially 0, but SRC_ADDR_L(n) after DMA (R)
MMIO[81E2h:0..7] - DMA Internal: initially 0, but DST_ADDR_L(n) after DMA (R)
MMIO[81E4h:0..7] - DMA Internal: initially 0, but SRC_ADDR_H(n) after DMA (R)
MMIO[81E6h:0..7] - DMA Internal: initially 0, but DST_ADDR_H(n) after DMA (R)
MMIO[81E8h..81FEh] - Unknown, fixed 0 (R)
Unknown/internal stuff.

```
_____________________________________ DMA Notes _____________________________________
```

STEP0, STEP1 and STEP2 specify the address stepping between elements for each dimension. Below is an example showing how it works.
With the source address configuration:

```
SRC_ADDR = 0
SIZE0 = 3
SIZE1 = 5
SIZE2 = 2
SRC_STEP0 = 2
SRC_STEP1 = 1
SRC_STEP2 = 7
```

The data transfer copies data from the following addresses:

```
<-------------------------size1------------------------->
<--size0--> <--size0--> <--size0--> <--size0--> <--size0-->
 0,  2,  4,  5,  7,  9, 10, 12, 14, 15, 17, 19, 20, 22, 24  <-- size2 (1st)
31, 33, 35, 36, 38, 40, 41, 43, 45, 46, 48, 50, 51, 53, 55  <-- size2 (2nd)
```

The 32bit double word mode also automatically aligns down the addresses to 32bit boundaries.

When the memory space is specified as 7 (AHBM), it performs data transfer from/to external ARM memory. The external memory has 8-bit addressing. Keep in mind that the STEP value is also added to the address as-is, so STEP = 1 means stepping 8-bit for FCRAM, while stepping 16-bit for DSP memory.
There are also more alignment requirement on the external memory address, but it is handled by AHBM.

DSi Teak MMIO[8200h] - Interrupt Control Unit (ICU)

MMIO[8200h] - ICU Interrupt Pending Flags (0=No, 1=IRQ Pending) (R)
MMIO[8202h] - ICU Interrupt Acknowledge (0=No change, 1=Set) (W)
MMIO[8204h] - ICU Interrupt Manual Trigger (0=Release, 1=Set) (R/W)
MMIO[8206h] - ICU Enable Interrupt routing to core interrupt 0 (R/W)
MMIO[8208h] - ICU Enable Interrupt routing to core interrupt 1 (R/W)
MMIO[820Ah] - ICU Enable Interrupt routing to core interrupt 2 (R/W)
MMIO[820Ch] - ICU Enable Interrupt routing to vectored interrupt (R/W)
MMIO[820Eh] - ICU Interrupt Trigger mode (0=Level, 1=Edge) (R/W)
MMIO[8210h] - ICU Interrupt Polarity (0=Normal, 1=Invert) (R/W)

```
0-8   No hardware IRQs (but can be used as Software IRQs via Manual Trigger)
9     Timer 1
10    Timer 0
11    BTDMP 0
12    BTDMP 1
13    SIO
14    APBP
15    DMA
```

MMIO[8212h+(0..15)*4] - ICU Vectored Interrupt 0..15 Address, bit16-31 (R/W)
MMIO[8214h+(0..15)*4] - ICU Vectored Interrupt 0..15 Address, bit0-15 (R/W)
```
  0-17  Address of interrupt handler for vectored interrupt 0..15
  18-30 Unused (0)
  31    Context switch on vectored interrupt 0..15 (0=Disable, 1=Enable)
```
Observe that upper/lower 16bit are arranged in BIG-ENDIAN style (unlike other registers with lower/upper 16bit).
Default values on reset are 0003FC00h.
Interrupt vector address for Interrupt 0..15, when using "vectored interrupt".
The core interrupts and NMI have fixed vector addresses.

MMIO[8252h] - ICU Interrupt Master Disable (0=Normal, 1=Off/undoc) (R/W)
```
  0-15  Master Disable for interrupt 0..15 (0=Normal, 1=Off, don't set pending)
```

MMIO[8254h] - Unknown, R/W mask 5555h (R/W)
MMIO[8256h] - Unknown, R/W mask 5555h (R/W)
Unknown.

Interrupt related CPU registers
DSi Teak CPU Control/Status Registers

teak exception vectors
```
  code:00000h    start (from reset or timer watchdog)
  code:00002h    trap_handler (trap/break) (from OCEM or Timers)
  code:00004h    nmi_handler (non-maskable interrupt) (from timer watchdog)
  code:00006h    int0_handler
  code:0000Eh    int1_handler
  code:00016h    int2_handler
  code:variable  vint_handler(s) (with context switch, instead of push/pop?)
```

Interrupt Behaviour
Pending flags are set somewhat as so:
```
  new_state=incoming_hw_signal             ;always 0 for interrupt 0..8
  if polarity=1 then new_state=new_state xor 1
  new_state=new_state OR manual_trigger    ;done AFTER above polarity invert
  new_state=new_state AND NOT master_disable
  if new_state=1 and old_state=0 then pending=1
  old_state=new_state
```
Acknowlege is required for clearing pending bits, this is required in both Egde- and Level-triggered modes (the difference is that Level-triggered mode will ignore the acknowledge if new_state is still on).
Manual Trigger won't appear in Pending until after two NOP opcodes (or similar delay).

IRQ-to-interrupt translator
The main job of ICU is to translate 16 IRQ signals to 4 processor interrupt signals (int0, int1, int2 and vint), specified by I0x, I1x, I2x and IVx registers. When an IRQ is signaled, ICU will generate interrupt signal and the processor starts to execute interrupt handling procedure if the core interrupt is enabled. The procedure is supposed check IPx reigster to know the exact IRQ index, and to set IAx registers to clear the IRQ signal.

Software interrupt
The processor can writes to ITx to generate a software interrupt manually. A use case for this in many 3DS games is that IRQ 0 is used as the reschedule procedure for multithreading. When a thread wants to yield, it triggers IRQ 0, which switches thread context and resume another thread.

DSi Teak MMIO[8280h] - Audio (Buffered Time Division Multiplexing

Port)

There are two BTDMP units. The weird name "Buffered Time Division Multiplexing Port" does essentially mean "FIFO with sample rate timer".

```
BTDMP 0 is used for Receive (microphone) and Transmit (audio out).
BTDMP 1 isn't actually used for anything.
```

MMIO[8280h/82A0h+(0..1)*80h] - BTDMP Receive/Transmit Control (R/W)

```
0-3    Unknown (0..Fh) (usually 0Dh/0Fh)
4-7    Unknown (0..Fh) (usually 00h)
8-11   Enable BTDMP Interrupt when non-zero (0=Off, AnyOther=On?)
12-15  Unknown (0..Fh) (usually 00h)
```

DSi sets this to 020Dh (receive) or 010Fh (transmit).

MMIO[8282h/82A2h+(0..1)*80h] - BTDMP Receive/Transmit Period (R/W)

```
0-2    Clock Divider? (0..7)    (usually 4)   (affects timing when bit13=1)
3-4    Unused (0)
5-12   Clock Divider? (0..FFh)  (usually 80h) (affects timing when bit13=1)
13     Clock Select   (0=ExternalDSiAudioClk, 1=InternalClkDivider?)
14     Unknown        (0..1)    (usually 0)
15     Unused (0)
```

Usually set to 1004h, which does select ExternalClk (32.73kHz or 47.61kHz, as selected in SNDEXCNT on ARM side; per 2x16bit stereo data).
Unknown how InternalClk works exactly. Increasing the "dividers" doesn't consistently increase FIFO transfer time; it seems as if the max time gets truncated somehow; maybe that's caused by InternalClk being ANDed with still running ExternalClk.

MMIO[8284h/82A4h+(0..1)*80h] - BTDMP Receive/Transmit Usually 0004h (R/W)

```
0-2    Unknown (0..7) (R/W)                                    (R/W)
3-4    Unused? (0)
5-11   Unknown (0..7Fh) (R/W)                                  (R/W)
12-15  Unused? (0)
```

Hmmmm, the write-ability of bit0-11 resembles the Period register...?

MMIO[8286h/82A6h+(0..1)*80h] - BTDMP Receive/Transmit Usually 0021h (R/W)

```
0-1    Unknown (0..3) (R/W)                                    (R/W)
2-4    Unused? (0)
5      Unknown/writeonly? (read=0) (code writes 1 here)        (W?)
6-15   Unused? (0)
```

MMIO[8288h/82A8h+(0..1)*80h] - BTDMP Receive/Transmit Usually 0000h'a (R/W)

```
0-12   Unknown (0..1FFFh) (R/W) (initially=1FFFh on reset)
13-15  Unused (0)
```

MMIO[828Ah/82AAh+(0..1)*80h] - BTDMP Receive/Transmit Usually 0000h'b (R/W)

```
0-11   Unknown (0..0FFFh) (R/W)
12-15  Unused (0)
```

MMIO[828Ch/82ACh+(0..1)*80h] - BTDMP Receive/Transmit Usually 0000h'c (R/W)

```
0-13   Unknown (0..3FFFh) (R/W)
14-15  Unused (0)
```

MMIO[828Eh/82AEh+(0..1)*80h] - BTDMP Receive/Transmit Usually unused'a (R/W)
MMIO[8290h/82B0h+(0..1)*80h] - BTDMP Receive/Transmit Usually unused'b (R/W)

```
0-15   Unknown (0..FFFFh) (R/W)
```

Some of the above unknown stuff might be... volume, bias, sample.size, mono/stereo, dummy-value-when-fifo-empty, beep generator, filter/equalizer, sample rate interpolation, whatever...?

MMIO[829Eh/82BEh+(0..1)*80h] - BTDMP Receive/Transmit Enable (R/W)

```
0-14   Unused (0)
15     Enable Transfer  (0=Off, 1=On, allow Transfer+IRQ's)

_____________________________ BTDMP FIFOs _____________________________
```

MMIO[82C0h/82C2h+(0..1)*80h] - BTDMP Receive/Transmit FIFO Status (R)

```
0      usually 0
1      If transfer ENABLED usually set, sometimes 0
2      If transfer ENABLED usually set
3      FIFO Full  (0=No, 1=Full, 16x16bit words)
4      FIFO Empty (0=No, 1=Empty, 0x16bit words)
5      If transfer ENABLED usually set
6      usually 0
7      For TX: gets set when FIFO contains ONE word?
8-15   usually 0
```

MMIO[82C4h/82C6h+(0..1)*80h] - BTDMP Receive/Transmit FIFO Data (R) and (W)

```
0-15      Signed 16bit audio sample.
```

FIFO size is 16x16bit. The data is stereo, ie. one needs to write 2x16bit for a full stereo sample (the write-order for left/right is still unknown).
Software seems to be directly writing to the FIFO (rather than using DMA). If DMA is also supported, that would probably require whatever special settings in the BTDMP and/or DMA registers (ie. so that the DMA knows when it needs to transfer new FIFO data).

MMIO[82C8h/82CAh+(0..1)*80h] - BTDMP Receive/Transmit FIFO Control (R/W)

```
0-1    Unknown     (0..3=?)                                          (R/W)
2      Flush FIFO  (0=No change, 1=Clear FIFO)                       (W)
3-15   Unused (0)
```

DSi Teak CPU Registers

36bit/40bit Accumulators: a0,a1,b0,b1

```
a0e:a0h:a0l (4:16:16 bits) = a0 (36bit)    ;TL2: 40bit (8:16:16)
a1e:a1h:a1l (4:16:16 bits) = a1 (36bit)    ;TL2: 40bit (8:16:16)
b0e:b0h:b0l (4:16:16 bits) = b0 (36bit)    ;TL2: 40bit (8:16:16)
b1e:b1h:b1l (4:16:16 bits) = b1 (36bit)    ;TL2: 40bit (8:16:16)
```

4bit snippets (bit32-35) of a0/a1 can be found in status registers (st0,st1). On TL2, the whole upper 8bit (bit32-39) of a0/a1/b0/b1 can be additionally accessed via push/pop (a0e,a1e,b0e,b1e).

Teak General Registers: r0,r1,r2,r3,r4,r5,r6,r7

```
r0     ;TL  ;16bit  ;\
r1     ;TL  ;16bit  ;
r2     ;TL  ;16bit  ; old TL1 registers
r3     ;TL  ;16bit  ;
r4     ;TL  ;16bit  ;
r5     ;TL  ;16bit  ;/
r6     ;TL2 ;16bit  ;<-- new TL2 register
r7     ;TL  ;16bit  ;<-- aka rb (with optional immediate, MemR7Imm)
```

32bit Multiply Result and 16bit Muliply Parameters: p0,p1, and x0,y0,x1,y1

```
x0     ;TL  ;16bit  ;-
y0     ;TL  ;16bit  ;-
x1     ;TL2 ;16bit  ;-
y1     ;TL2 ;16bit  ;-
p0     ;TL  ;33bit! ;\Px   ;TL2: 33bit p0e:p0 ?  ;TL1: 32bit?
```

```
  p1     ;TL2 ;33bit! ;/     ;TL2: 33bit p1e:p1 ?  ;TL1: N/A
  p0h    ;TL  ;16bit  ; ;<-- aka ph ;<-- called "p0" (aka "p") in "RegisterP0"
```
The "load ps" and "load ps01" opcodes allow to specify a multiply shifter, this is useful when dealing with signed/unsigned parameters:
```
  Unsigned = Unsigned * Unsigned       ;use shift 0
  Unsigned = Unsigned * Signed         ;use shift +1
  Unsigned = Signed * Signed           ;use shift +2
  Signed = Unsigned * Unsigned         ;use shift -1
  Signed = Unsigned * Signed           ;use shift 0
  Signed = Signed * Signed             ;use shift +1
```

TeakLite Misc
```
  pc     ;TL  ;18bit! ;-program counter (TL2: 18bit, TL1: 16bit)
  sp     ;TL  ;16bit  ;-stack pointer (decreasing on push/call)
  sv     ;TL  ;16bit  ;-shift value (negative=right) (for shift-by-register)
  mixp   ;TL  ;16bit  ;-related to min/max/mind/maxd
  lc     ;TL  ;16bit  ;-Loop Counter (of block repeat)
  repc   ;TL  ;16bit  ;-Repeat Counter (for "rep" opcode)
  dvm    ;TL  ;16bit  ;-Data Value Match (OCEM data breakpoints) (and for trap)
```

TeakLiteII Misc: vtr0,vtr1,prpage
```
  vtr0   ;TL2 16bit   ;\related to vtrshr,vtrmov,vtrclr
  vtr1   ;TL2 16bit   ;/(saved C/C1 carry flags for Viterby decoding)
  prpage ;TL2 4bit    ;-??? (bit0-3 used/dangerous, bit4-15 always 0)
```
vtr0,vtr1 are related to vtrshr,vtrmov,vtrclr opcodes (and multi-function opcodes with "vtrshr" suffix).
prpage isn't used by existing DSi code, setting the four write-able bits to nonzero seems to screw-up opcode fetching, causing code to crash (unless one of the next 1-2 prefetched opcodes restores prpage=0, which causes opcode fetching to recover; after skipping some following prefetched opcodes, until prpage=0 is applied). Maybe it's related to code access rights or waitstates... it doesn't seem to be related to upper 2bit of the 18bit program counter (prpage is zero even when executing code above address 0FFFFh).

Old Control/Status registers (TeakLite): st0,st1,st2,icr
New Control/Status registers (TeakLiteII): stt0,stt1,stt2,mod0,mod1,mod2,mod3
DSi Teak CPU Control/Status Registers

Address Config (TeakLiteII): ar0,ar1,arp0,arp1,arp2,arp3
Address Step/Modulo: cfgi,cfgj (and TL2 stepi0,stepj0)
DSi Teak CPU Address Config/Step/Modulo

User-defined registers (optional off-core): ext0,ext1,ext2,ext3
The four ext registers are intended for custom hardware extensions (where they could be used as I/O ports, with faster & more direct access than memory mapped I/O).
```
  ext0   ;TL  ;16bit
  ext1   ;TL  ;16bit
  ext2   ;TL  ;16bit
  ext3   ;TL  ;16bit
```
In the DSi, the four register do exist (they are fully read/write-able), but unknown if they do have any special functions - or if they are just general-purpose data registers (existing DSi software isn't using the ext registers, and hardware is solely accessed via memory mapped I/O).

Bitfields for Control/Status registers and cfgi/cfgj registers
```
  page   ;TL  ;8bit "load" st1.bit0-7   (page for MemImm8)    ;aka "lpg"
  ps     ;TL  ;2bit "load" st1.bit10-11 (product shifter for multiply?)
  ps01   ;TL2 ;4bit "load" mod0...?     (maybe separate "ps" for p0 and p1 ?)
  movpd  ;TL2 ;2bit "load" stt2.bit6-7  (page for reading DATA from ProgMem)
  modi   ;TL  ;9bit "load" cfgi.bit7-15 =imm9
  modj   ;TL  ;9bit "load" cfgj.bit7-15 =imm9
  stepi  ;TL  ;7bit "load" cfgi.bit0-6  =imm7
  stepj  ;TL  ;7bit "load" cfgj.bit0-6  =imm7
```

Shadow Registers
Some registers (or in case of st0-st2: fractions thereof) exist as "shadows"... related to "cntx", "swap", "banke" (and maybe "bankr"?) opcodes, and "reti/retid" opcodes with "context" suffix, and interrupts with context switching enabled.

```
  st0  bit0,2-11                        ;\control/status (cntx)
  st1  bit10-11 (and "swap": bit0-7)    ; (TL2: probably also SttMod)
  st2  bit0-7                           ;/
  a0 <--> b0    manualswap only?        ;\accumulators (swap)
  a1 <--> b1    autoswapped?            ;/
  r0 <--> r0b                           ;\
  r1 <--> r1b                           ;
  r4 <--> r4b                           ; BankFlags (banke)
  r7 <--> r7b     ;TL2                  ;
  cfgi <--> cfgib                       ;
  cfgj <--> cfgjb ;TL2                  ;/
  Ar,Arp <--> ?   ;TL2                  ;-? (bankr and/or cntx)
```

Registers b0/b1 can be used as normal opcode operands, the other shadow registers are used only when doing bank/cntx stuff.

Suffix codes: dmod,dmodi,dmodj,dmodij,context,eu,dbrv,ebrv,s,r
Non-register assembler keywords.

```
  dmod   ;TL  ;suffix ;\
  dmodi  ;TL2 ;suffix ;
  dmodj  ;TL2 ;suffix ;
  dmodij ;TL2 ;suffix ;/
  context;TL  ;suffix ;<-- (related to "cntx")
  eu     ;TL  ;suffix ;<-- (aka "Axheu", now "Axh,eu")
  dbrv   ;TL2 ;suffix ;\for "bitrev"
  ebrv   ;TL2 ;suffix ;/
  s      ;TL  ;suffix ;\param for "cntx" opcode  ;"s" also for opcode 88D1h
  r      ;TL  ;suffix ;/
```

Condition codes: true,eq,neq,gt,ge,lt,le,nn,c,v,e,l,nr,niu0,iu0,iu1
The 16 condition codes can be used for all opcodes with "Cond" operand, whereas "true" can be omitted (as it means always/non-conditional), the four conditions "gt,ge,lt,le" can be also used with "min/max/maxd/cbs" opcodes.

Old pre-TeakLiteII keyword names (renamed in TeakLiteII)

```
  TL:   x    y    p    ph   rb   lpg   a0heu   a1heu
  TL2:  x0   y0   p0   p0h  r7   page  a0h,eu  a1h,eu
```

DSi Teak CPU Control/Status Registers

There are two sets of Control/Status registers

```
  Old registers (for TeakLite):   st0/st1/st2, and icr
  New registers (for TeakLiteII): stt0/stt1/stt2, and mod0/mod1/mod2/mod3
```

The new registers do contain only a few new bits, apart from that they are basically same as the old registers (with the old bits rearranged to different locations in the new registers).
The old registers do still exist on TeakLiteII, so one could use either old or new registers (all reads/writes will be mirrored to both register sets).
However, there are a few cases where writing the old registers may smash bits in new registers (writing the old "limit" bit will change BOTH of the new "limit" bits, changing "ps" versus "ps01" may also involve strange effects, and... changing "a0e/a1e" bits seems to have "weird" effects on a0l/a0h/a1l/a1h, or maybe that's some "wanted" saturation effect?).

CPU Flags (for Cond opcodes)

CPU Flags are stored in st0 register (with mirrors in stt0/stt1). The flags can be used for conditional opcodes (with "Cond" operand). According to TeakLite datasheet, the flags are affected somewhat like this:

```
ZMNVCEL- add, addh, addl, cmp, cmpu, sub, subh, subl, inc, dec, neg
ZMNVCEL- maa, maasu, mac, macsu, macus, macuu, msu, sqra, rnd, pacr, movr
ZMN-C--- or
ZM--C--- addv, cmpv, subv, and
ZMN--E-- clr, clrr, copy, divs, swap, not, xor
ZMN--0L- lim
ZMNVCELR norm
ZMN-CE-- rol, ror
ZMN-CE-- movs, movsi, shfc, shfi, shl, shl4, shr, shr4  ;for logical shift
ZMNVCEL- movs, movsi, shfc, shfi, shl, shl4, shr, shr4  ;for arithmetic shift
ZMN--E-- mov, movp, pop        ;when dst=ac,bc (whut?)         ;\
xxxxxxxx mov, movp, pop        ;when dst=st0                   ; mov etc.
------L- mov, push             ;when src=aXL,aXH,bXL,bXH       ;
-------- mov, movp, pop, push  ;when src/dst neither of above  ;/
ZMN--E-- cntx s  ;store shadows (new flags for a1)             ;\cntx
ZMNVCELR cntx r  ;restore shadows (old flags)                  ;/
ZM------ set, rst, chng
Z------- tst0, tst1, tstb
-M------ max, maxd, min
-------R modr
-------- mpy, mpyi, mpysu, sqr, exp
-------- banke, dint, eint, load, nop, bkrep, rep, break, trap, movd
-------- br, brr, call, calla, callr, ret, retd, reti, retid, rets
```

Flags for new TL2 opcodes aren't officially documented; some might follow the above rules (eg. the new "r6" register should act as old "r0-r5"), but other new opcodes might do this or that.

Old registers (TeakLite)

st0 - Old TL1 Status/Control Register st0

```
0     SAT  R/W Saturation Mode  (0=Off, 1=Saturate "Ax to data") ;mod0.0
1     IE   R/W Interrupt Enable (0=Disable, 1=Enable) ;dint/eint ;mod3.7
2     IM0  R/W Interrupt INT0 Mask (0=Disable, 1=Enable if IE=1) ;mod3.8
3     IM1  R/W Interrupt INT1 Mask (0=Disable, 1=Enable if IE=1) ;mod3.9
4     R    R/W Flag: rN is Zero   ;see Cond nr                   ;stt1.4
5     L    R/W Flag: Limit        ;see Cond l      ;L=(LM or VL) ;stt0.0+1
6     E    R/W Flag: Extension    ;see Cond e                    ;stt0.2
7     C    R/W Flag: Carry        ;see Cond c                    ;stt0.3
8     V    R/W Flag: Overflow     ;see Cond v                    ;stt0.4
9     N    R/W Flag: Normalized   ;see Cond nn                   ;stt0.5
10    M    R/W Flag: Minus        ;see Cond gt,ge,lt,le          ;stt0.6
11    Z    R/W Flag: Zero         ;see Cond eq,neq,gt,le         ;stt0.7
12-15 a0e  R/W Accumulator 0 Extension Bits                      ;a0.32-35
```

st1 - Old TL1 Status/Control Register st1

```
0-7   PAGE R/W Data Memory Page (for MemImm8) (see "load page")  ;mod1.0-7
8-9   -    -   Reserved (read: always set)                      ;-
10-11 PS   R/W Product Shifter for P0 (see "load ps")(multiply?) ;mod0.10-11
                 (0=No Shift, 1=SHR1, 2=SHL1, 3=SHL2)
12-15 a1e  R/W Accumulator 1 Extension Bits                      ;a1.32-35
```

st2 - Old TL1 Status/Control Register st2

```
0-3   MDn  R/W Enable cfgi.modi modulo for R0..R3 (0=Off, 1=On)  ;mod2.0-3
4-5   MDn  R/W Enable cfgj.modj modulo for R4..R5 (0=Off, 1=On)  ;mod2.4-5
6     IM2  R/W Interrupt INT2 Mask (0=Disable, 1=Enable if IE=1) ;mod3.10
7     S    R/W Shift Mode (0=Arithmetic, 1=Logic)                ;mod0.7
8     OU0  R/W OUSER0 User Output Pin                            ;mod0.8
9     OU1  R/W OUSER1 User Output Pin                            ;mod0.9
10    IU0  R   IUSER0 User Input Pin (zero)  ;see Cond iu0,niu0  ;stt1.??
11    IU1  R   IUSER1 User Input Pin (zero)  ;see Cond iu1       ;stt1.??
12    -    -   Reserved (read: always set)                       ;-
13    IP2  R   Interrupt Pending INT2 (0=No, 1=IRQ)              ;stt2.2
```

```
  14    IP0  R   Interrupt Pending INT0 (0=No, 1=IRQ)             ;stt2.0
  15    IP1  R   Interrupt Pending INT1 (0=No, 1=IRQ)             ;stt2.1
```

icr - Old TL1 Interrupt Context and Repeat Nesting

```
  0     NMIC R/W NMI Context switching enable  (0=Off, 1=On)      ;mod3.0
  1     IC0  R/W INT0 Context switching enable (0=Off, 1=On)      ;mod3.1
  2     IC1  R/W INT1 Context switching enable (0=Off, 1=On)      ;mod3.2
  3     IC2  R/W INT2 Context switching enable (0=Off, 1=On)      ;mod3.3
  4     LP   R   InLoop (when inside one or more "bkrep" loops)   ;stt2.15
  5-7   BCn  R   Block repeat nest. counter ;see "bkrep"          ;stt2.12-14
  8-15  -    -   Reserved (read: always set)                      ;-
```

_____ New registers (TeakLiteII) _____

stt0 - New TL2 Status/Control Register stt0 (CPU Flags)

```
  0     LM   R/W Flag: Limit, set if saturation has/had occured   ;st0.5
  1     VL   R/W Flag: LatchedV, set if overflow has/had occurred ;st0.5, too
  2     E    R/W Flag: Extension    ;see Cond e                   ;st0.6
  3     C    R/W Flag: Carry        ;see Cond c                   ;st0.7
  4     V    R/W Flag: Overflow     ;see Cond v                   ;st0.8
  5     N    R/W Flag: Normalized   ;see Cond nn                  ;st0.9
  6     M    R/W Flag: Minus        ;see Cond gt,ge,lt,le         ;st0.10
  7     Z    R/W Flag: Zero         ;see Cond eq,neq,gt,le        ;st0.11
  8-10  -    -   Unknown (reads as zero)
  11    C1   R/W Flag: Carry1 (2nd carry, for dual-operation opcodes)
  12-15 -    -   Unknown (reads as zero)
```

stt1 - New TL2 Status/Control Register stt1 (whatever)

```
  0-3   -    -   Unknown (reads as zero)
  4     R    R/W Flag: rN is Zero   ;see Cond nr                  ;st0.4
  5-13  -    -   Unknown (reads as zero)  (IU1 and IU0 should be here!)
  14    P0E  R/W Upper bit of 33bit P0 register  ;\shifted-in on       ;p0.32
  15    P1E  R/W Upper bit of 33bit P1 register  ;/arith right shifts  ;p1.32
```

Note: bit14/bit15 are automatically sign-expanded when moving data to p0/p0h/p1.

stt2 - New TL2 Status/Control Register stt2 (Interrupt/ProgBank/Bkrep)

```
  0     IP0  R   Interrupt Pending INT0 (0=No, 1=IRQ)             ;st2.14
  1     IP1  R   Interrupt Pending INT1 (0=No, 1=IRQ)             ;st2.15
  2     IP2  R   Interrupt Pending INT2 (0=No, 1=IRQ)             ;st2.13
  3     IPV  R   Interrupt Pending VINT                           ;-
  4-5   -    -   Unknown (reads as zero)                         ;-
  6-7  PCMhi R/W Program Memory Bank (for ProgMemRn/ProgMemAxl) ("load movpd")
  8-11  -    -   Unknown (reads as zero)                         ;-
  12-14 BCn  R   Block repeat nest. counter ;see "bkrep"         ;icr.5-7
  15    LP   R   InLoop (when inside one or more "bkrep" loops)  ;icr.4
```

mod0 - New TL2 Status/Control Register mod0 (Misc)

```
  0     SAT  R/W Saturation Mode (0=Off, 1=Saturate "Ax to data"?) ;st0.0
  1     SATA R/W Saturation Mode on store (0=Off, 1="(Ax op data) to Ax"?)
  2     ?    R   Unknown (reads as one)
  3     -    -   Unknown (reads as zero)
  4     -    -   Unknown (reads as zero)
  5-6   HWM  R/W Halfword Multiply    ... Modify y0 (and y1?)
                  0=read y0/y1 directly (full 16bit words)
                  1=Takes y0>>8 and y1>>8 (logic shift)
                  2=Takes y0&0xFF and y1&0xFF
                  3=Takes y0>>8 and y1&&0xFF
  7     S    R/W Shift Mode (0=Arithmetic, 1=Logic)               ;st2.7
  8     OU0  R/W OUSER0 User Output Pin                           ;st2.8
  9     OU1  R/W OUSER1 User Output Pin                           ;st2.9
  10-11 PS0  R/W Product Shifter for P0 (see "load ps")(multiply?) ;st1.10-11
  12    -    -   Unknown (reads as zero)
```

```
  13-14 PS1  R/W Product Shifter for P1 (see "load ps")(multiply?)
  15    -    -   Unknown (reads as zero)
```

mod1 - New TL2 Status/Control Register mod1 (Data Page)

```
  0-7   PAGE R/W Data Memory Page (for MemImm8) (see "load page")  ;st1.0-7
  8-11  -    -   Unknown (reads as zero)
  12   STP16 R/W banke opcode (0=exchange cfgi/cfgj, 1=cfgi/cfgj+stepi0/stepj0)
                 1=use stepi0/j0 instead of stepi/j for stepping Rn registers
  13    CMD  R/W Change Modulo mode (0=New TL2 style, 1=TL1 style)
  14    EPI  R/W Unknown (1=Set R3=0 after any "modr R3" or "access[R3]"?)
  15    EPJ  R/W Unknown (1=Set R7=0 after any "modr R7" or "access[R7]"?)
```

mod2 - New TL2 Status/Control Register mod2 (Modulo Enable)

```
  0-3   MDn  R/W Enable cfgi.modi modulo for R0..R3 (0=Off, 1=On)  ;st2.0-3
  4-5   MDn  R/W Enable cfgj.modj modulo for R4..R5 (0=Off, 1=On)  ;st2.4-5
  6-7   MDn  R/W Enable cfgj.modj modulo for R6..R7 (0=Off, 1=On) ;TL2 only
  8-11  BRn  R/W Step +s for R0..R3 (0=cfgi.stepi, 1=stepi0)
  12-15 BRn  R/W Step +s for R4..R7 (0=cfgj.stepi, 1=stepj0)
```

XXX... bit8-9 seem to mess up my code (that uses r0/r1, but only with +0 step).
"When BRn=1, memory access through Rn will use the bit-reversed value of Rn as the address. Note that this also implies that stepi0/j0 will be used, regardless of what STP16 says."

mod3 - New TL2 Status/Control Register mod3 (Interrupt Control)

```
  0     NMIC R/W NMI Context switching enable  (0=Off, 1=On)       ;icr.0
  1     IC0  R/W INT0 Context switching enable (0=Off, 1=On)       ;icr.1
  2     IC1  R/W INT1 Context switching enable (0=Off, 1=On)       ;icr.2
  3     IC2  R/W INT2 Context switching enable (0=Off, 1=On)       ;icr.3
  4     OU2  R/W Unknown (R/W)
  5     OU3  R/W Unknown (R/W)
  6     OU4  ?   ---DANGER BIT--- (1=hangs/crashes when set)
  7     IE   R/W Interrupt Enable (0=Disable, 1=Enable) ;dint/eint ;st0.1
  8     IM0  R/W Interrupt INT0 Mask (0=Disable, 1=Enable if IE=1) ;st0.2
  9     IM1  R/W Interrupt INT1 Mask (0=Disable, 1=Enable if IE=1) ;st0.3
  10    IM2  R/W Interrupt INT2 Mask (0=Disable, 1=Enable if IE=1) ;st2.6
  11    IMV  R/W Interrupt VINT Mask (0=Disable, 1=Enable if IE=1?)
  12    -    -   Unknown (reads as zero)
  13   CCNTA R/W Unknown (R/W)
  14    CPC  R/W Stack word order for PC on call/ret (0=Normal, 1=Reversed)
  15    CREP R/W Unknown (R/W)
```

Bit14=0: push lowword then push highword on call; pop highword then pop lowword on ret.

DSi Teak CPU Address Config/Step/Modulo

```
_____________________________ Address Config _____________________________
```

These registers allow to reconfigure opcodes. For example, opcodes with operand MemR45 can normally use only R4/R5 operands; if desired that can be changed to anything else like R3/R0. The selection if Offset/Step operands can be changed in similar way.

```
  Unknown which settings affect which opcodes exactly.
```

The DSi Sound code is using a default configuration for most of its code, but it does also have some functions with alternate configurations.

ar0/ar1

```
  0-2   R/W PM1/PM3 Post Modify Step    (0..7 = +0,+1,-1,+s,+2,-2,+2,-2)
  3-4   R/W CS1/CS3 Offset              (0..3 = +0,+1,-1,-1)
  5-7   R/W PM0/PM2 Post Modify Step    (0..7 = +0,+1,-1,+s,+2,-2,+2,-2)
  8-9   R/W CS0/CS2 Offset              (0..3 = +0,+1,-1,-1)
  10-12 R/W RN1/RN3 Register            (0..7 = R0..R7)
  13-15 R/W RN0/RN2 Register            (0..7 = R0..R7)
```

arp0/arp1/arp2/arp3

```
  0-2   R/W PIn     Post Modify Step I  (0..7 = +0,+1,-1,+s,+2,-2,+2,-2)
  3-4   R/W CIn     Offset I            (0..3 = +0,+1,-1,-1)
  5-7   R/W PJn     Post Modify Step J  (0..7 = +0,+1,-1,+s,+2,-2,+2,-2)
  8-9   R/W CJn     Offset J            (0..3 = +0,+1,-1,-1)
  10-11 R/W RIn     Register I          (0..3 = R0..R3)
  12    -   -       Unused              (always zero)
  13-14 R/W RJn     Register J          (0..3 = R4..R7)
  15    -   -       Unused              (always zero)

 _____________________________ Step/Modulo _____________________________
```

cfgi - Step and Mod I (for R0..R3)
cfgj - Step and Mod J (for R4..R7)

```
  0-6   stepi/stepj  (7bit) (see "load stepi/stepj")  ;step "Rn+s" ?
  7-15  modi/modj    (9bit) (see "load modi/modj")
```

The modulos can be enabled in Control/Status registers. Some opcodes do also allow to disable modulos via "dmod" suffix.
On TL2, the above 7bit stepi/stepj can be optionally replaced by new 16bit stepi0/stepj0 registers (via flags in mod2 register).

stepi0 ;TL2 16bit
stepj0 ;TL2 16bit

```
  0-16  stepi0/stepj0
```

more steps, probably for "modr" with "+s0" (stepII2D2S0)
and for STP16 and BRn?

DSi TeakLite II Instruction Set Encoding

The opcodes are 16bits wide (some followed by an additional 16bit parameter word, namely those with "@16" operands). The encoding is very messy (fixed opcode bits randomly mixed/interleaved with variable parameter bits, and with new TL2 opcodes squeezed in formerly unused locations), making it pretty much impossible to decode that unpleasant stuff by software/logic.
The only reasonable decoding way is using a huge table with 65536 entries (which could be generated temporarily from the information in below table, using the Base number plus all variable bit combinations, for example, "6100h TL mov MemImm8@0, Ab@11" has variable bits in bit0-7 and bit11-12, so the opcode would be mapped at 6100h-61FFh, 6900h-69FFh, 7100h-71FFh, 7900h-79FFh).

TeakLite I (TL) and TeakLite II (TL2) Opcodes

```
  Base  Ver Opcode (with parameter bits located at @bitnumber and up)
  D4FBh TL  add  MemImm16@16, Ax@8
  A600h TL  add  MemImm8@0, Ax@8
  86C0h TL  add  Imm16@16, Ax@8
  C600h TL  add  Imm8u@0, Ax@8
  D4DBh TL  add  MemR7Imm16@16, Ax@8
  4600h TL  add  MemR7Imm7s@0, Ax@8
  8680h TL  add  MemRn@0, Ax@8 || Rn@0stepZIDS@3
  86A0h TL  add  RegisterP0@0, Ax@8
  D2DAh TL2 add  Ab@10, Bx@0
  5DF0h TL2 add  Bx@1, Ax@0
  9070h TL2 add  MemR01@8, sv, Abh@2 || sub MemR01@8offsZI@0, sv, Abl@2
             || mov Abl@2, MemR45@8 || R01@8stepII2@0, R45@8stepII2@1
  5DB0h TL2 add  MemR04@1, sv, Abh@2 || sub MemR04@1offsZI@0, sv, Abl@2
             || R04@1stepII2@0
  6F80h TL2 add  MemR45@2, MemR01@2, Abh@3
             || add MemR45@2offsZI@1, MemR01@2offsZI@0, Abl@3
```

```
           || R01@2stepII2@0, R45@2stepII2@1
6FA0h TL2 add  MemR45@2, MemR01@2, Abh@3
           || sub MemR45@2offsZI@1, MemR01@2offsZI@0, Abl@3
           || R01@2stepII2@0, R45@2stepII2@1
5E30h TL2 add  MemR45@8, sv, Abh@2 || sub MemR45@8offsZI@1, sv, Abl@2
           || mov Abl@2, MemR01@8 || R01@8stepII2@0, R45@8stepII2@1
5DC0h TL2 add  p0, p1, Ab@2
D782h TL2 add  p1, Ax@0
5DF8h TL2 add  Px@1, Bx@0
D38Bh TL2 add  r6, Ax@4
4590h TL2 add3 p0, p1, Ab@2
4592h TL2 add3a p0, p1, Ab@2
4593h TL2 add3aa p0, p1, Ab@2
5DC1h TL2 adda p0, p1, Ab@2
B200h TL  addh MemImm8@0, Ax@8
9280h TL  addh MemRn@0, Ax@8 || Rn@0stepZIDS@3
92A0h TL  addh Register@0, Ax@8
9464h TL2 addh r6, Ax@0
90E0h TL2 addhp MemR0425@2, Px@4, Ax@8 || R0425@2stepII2D2S@0 ;p=ProgMem? Px?
B400h TL  addl MemImm8@0, Ax@8
9480h TL  addl MemRn@0, Ax@8 || Rn@0stepZIDS@3
94A0h TL  addl Register@0, Ax@8
9466h TL2 addl r6, Ax@0
906Ch TL2 addsub  p0, p1, Ab@0
49C2h TL2 addsub  p1, p0, Ab@4
916Ch TL2 addsuba p0, p1, Ab@0
49C3h TL2 addsuba p1, p0, Ab@4
E700h TL  addv Imm16@16, MemImm8@0
86E0h TL  addv Imm16@16, MemRn@0 || Rn@0stepZIDS@3
87E0h TL  addv Imm16@16, Register@0
47BBh TL2 addv Imm16@16, r6
D4F9h TL  and  MemImm16@16, Ax@8
A200h TL  and  MemImm8@0, Ax@8
82C0h TL  and  Imm16@16, Ax@8
C200h TL  and  Imm8u@0, Ax@8
D4D9h TL  and  MemR7Imm16@16, Ax@8
4200h TL  and  MemR7Imm7s@0, Ax@8
8280h TL  and  MemRn@0, Ax@8 || Rn@0stepZIDS@3
82A0h TL  and  RegisterP0@0, Ax@8
6770h TL2 and  Ab@2, Ab@0, Ax@12               ;TL2 only
D389h TL2 and  r6, Ax@4
4B80h TL  banke BankFlags6@0  ;{r0}{,r1}{,r4}{,cfgi}{,r7}{,cfgj}
8CDFh TL2 bankr       ;without operand ?
8CDCh TL2 bankr Ar@0
8CD0h TL2 bankr Ar@2, Arp@0
8CD8h TL2 bankr Arp@0
5EB8h TL2 bitrev Rn@0
D7E8h TL2 bitrev Rn@0, dbrv
D7E0h TL2 bitrev Rn@0, ebrv
5C00h TL  bkrep NoReverse, Imm8u@0, Address16@16
5D00h TL  bkrep NoReverse, Register@0, Address18@16and5
8FDCh TL2 bkrep NoReverse, r6, Address18@16and0
DA9Ch TL2 bkreprst MemR0425@0
5F48h TL2 bkreprst MemSp, Unused2@0
DADCh TL2 bkrepsto MemR0425@0, Unused1@10
9468h TL2 bkrepsto MemSp, Unused3@0
4180h TL  br   Address18@16and4, Cond@0
D3C0h TL  break               ;break
5000h TL  brr  RelAddr7@4, Cond@0
41C0h TL  call Address18@16and4, Cond@0
D480h TL  calla Axl@8
D381h TL2 calla Ax@4
1000h TL  callr RelAddr7@4, Cond@0
9068h TL2 cbs  Axh@0, Axh@not0, r0, ge
9168h TL2 cbs  Axh@0, Axh@not0, r0, gt
```

```
D49Eh TL2 cbs  Axh@8, Bxh@5, r0, ge
D49Fh TL2 cbs  Axh@8, Bxh@5, r0, gt
D5C0h TL2 cbs  MemR01@2, MemR45@2, ge || R01@2stepII2@0, R45@2stepII2@1
D5C8h TL2 cbs  MemR01@2, MemR45@2, gt || R01@2stepII2@0, R45@2stepII2@1
E500h TL  chng Imm16@16, MemImm8@0
84E0h TL  chng Imm16@16, MemRn@0 || Rn@0stepZIDS@3
85E0h TL  chng Imm16@16, Register@0
47BAh TL2 chng Imm16@16, r6
0038h TL2 chng Imm16@16, SttMod@0
6760h TL  clr  Implied ConstZero, Ax@12, Cond@0  ;aX=0
6F60h TL  clr  Implied ConstZero, Bx@12, Cond@0  ;bX=0
8ED0h TL2 clr  Implied ConstZero, Ab@2, Ab@0
5DFEh TL2 clrp p0
5DFFh TL2 clrp p0, p1
5DFDh TL2 clrp p1
67C0h TL  clrr Implied Const8000h, Ax@12, Cond@0 ;aX=8000h
6F70h TL2 clrr Implied Const8000h, Bx@12, Cond@0 ;bX=8000h
8DD0h TL2 clrr Implied Const8000h, Ab@2, Ab@0
D4FEh TL  cmp  MemImm16@16, Ax@8
AC00h TL  cmp  MemImm8@0, Ax@8
8CC0h TL  cmp  Imm16@16, Ax@8
CC00h TL  cmp  Imm8u@0, Ax@8
D4DEh TL  cmp  MemR7Imm16@16, Ax@8
4C00h TL  cmp  MemR7Imm7s@0, Ax@8
8C80h TL  cmp  MemRn@0, Ax@8 || Rn@0stepZIDS@3
8CA0h TL  cmp  RegisterP0@0, Ax@8
4D8Ch TL2 cmp  Ax@1, Bx@0
D483h TL2 cmp  b0, b1
D583h TL2 cmp  b1, b0
DA9Ah TL2 cmp  Bx@10, Ax@0
8B63h TL2 cmp  p1, Ax@4
D38Eh TL2 cmp  r6, Ax@4
BE00h TL  cmpu MemImm8@0, Ax@8
9E80h TL  cmpu MemRn@0, Ax@8 || Rn@0stepZIDS@3
9EA0h TL  cmpu Register@0, Ax@8
8A63h TL2 cmpu r6, Ax@3
ED00h TL  cmpv Imm16@16, MemImm8@0
8CE0h TL  cmpv Imm16@16, MemRn@0 || Rn@0stepZIDS@3
8DE0h TL  cmpv Imm16@16, Register@0
47BEh TL2 cmpv Imm16@16, r6
D390h TL  cntx r  ;restore shadows
D380h TL  cntx s  ;store shadows
67F0h TL  copy Implied Ax@not12,   Ax@12, Cond@0 ;aX=aY
67E0h TL  dec  Implied Const1,     Ax@12, Cond@0 ;aX=aX-1
43C0h TL  dint        ;IE=0, interrupt disable
0E00h TL  divs MemImm8@0, Ax@8
4380h TL  eint        ;IE=1, interrupt enable
9460h TL  exp  Bx@0, Implied sv
9060h TL  exp  Bx@0, Implied sv, Ax@8
9C40h TL  exp  MemRn@0, Implied sv || Rn@0stepZIDS@3
9840h TL  exp  MemRn@0, Implied sv, Ax@8 || Rn@0stepZIDS@3
9040h TL  exp  RegisterP0@0, Implied sv, Ax@8
9440h TL  exp  RegisterP0@0, Implied sv
D7C1h TL2 exp  r6, Implied sv
D382h TL2 exp  r6, Implied sv, Ax@4
67D0h TL  inc  Implied Const1,     Ax@12, Cond@0 ;aX=aX+1
49C0h TL  lim  a0     ;aka a0,a0
49D0h TL  lim  a0, a1
49F0h TL  lim  a1     ;aka a1,a1
49E0h TL  lim  a1, a0
4D80h TL  load Imm2u@0, ps               ;st1.bit11-10=imm2
DB80h TL  load Imm7s@0, stepi            ;cfgi.LSB=imm7
DF80h TL  load Imm7s@0, stepj            ;cfgj.LSB=imm7
0400h TL  load Imm8u@0, page             ;st1.LSBs=imm8 ;aka "lpg"
0200h TL  load Imm9u@0, modi             ;cfgi.MSB=imm9
```

```
0A00h TL  load Imm9u@0, modj             ;cfgj.MSB=imm9
D7D8h TL2 load Imm2u@1, movpd, Unused1@0 ;stt2.bit6.7 (page for ProgMem)
0010h TL2 load Imm4u@0, ps01             ;mod0.bit10-11,13-14 and st1.10-11 ?
D400h TL  maa  MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8400h TL  maa  MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
8420h TL  maa  y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8440h TL  maa  y0, Register@0, Ax@11
E400h TL  maa  y0, MemImm8@0, Ax@11
5EA8h TL2 maa  y0, r6, Ax@0
D700h TL  maasu MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8700h TL  maasu MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
8720h TL  maasu y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8740h TL  maasu y0, Register@0, Ax@11
5EAEh TL2 maasu y0, r6, Ax@0
D200h TL  mac  MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8200h TL  mac  MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
8220h TL  mac  y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8240h TL  mac  y0, Register@0, Ax@11
E200h TL  mac  y0, MemImm8@0, Ax@11
5EA4h TL2 mac  y0, r6, Ax@0
4D84h TL2 mac  y0, x1, Ax@1, Unused1@0
5E28h TL2 mac1 MemR45@2, MemR01@2, Ax@8 || R01@2stepII2@0, R45@2stepII2@1
D600h TL  macsu MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8600h TL  macsu MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
E600h TL  macsu y0, MemImm8@0, Ax@11
8620h TL  macsu y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8640h TL  macsu y0, Register@0, Ax@11
5EACh TL2 macsu y0, r6, Ax@0
D300h TL  macus MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8300h TL  macus MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
8320h TL  macus y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8340h TL  macus y0, Register@0, Ax@11
5EA6h TL2 macus y0, r6, Ax@0
D500h TL  macuu MemR45@2, MemR0123@0, Ax@11
           || R0123@0stepZIDS@3, R45@2stepZIDS@5
8500h TL  macuu MemRn@0, Imm16@16, Ax@11 || Rn@0stepZIDS@3
8520h TL  macuu y0, MemRn@0, Ax@11 || Rn@0stepZIDS@3
8540h TL  macuu y0, Register@0, Ax@11
5EAAh TL2 macuu y0, r6, Ax@0
8460h TL  max  NoReverse, Ax@8, Implied Ax@not8, Bogus MemR0, ge,
           Implied mixp, Implied r0 || R0stepZIDS@3   ;when aY >= aX
8660h TL  max  NoReverse, Ax@8, Implied Ax@not8, Bogus MemR0, gt,
           Implied mixp, Implied r0 || R0stepZIDS@3   ;when aY > aX
5E21h TL2 max  a0h, a1h || max a0l, a1l || vtrshr
5F21h TL2 max  a1h, a0h || max a1l, a0l || vtrshr
D784h TL2 max  Axh@1, Bxh@0 || max Axl@1, Bxl@0 || vtrshr
4A40h TL2 max  Axh@3, Bxh@4 || max Axl@3, Bxl@4 || mov Axl@not3, MemR04@1
           || vtrshr || R04@1stepII2@0
4A44h TL2 max  Axh@3, Bxh@4 || max Axl@3, Bxl@4 || mov Axh@not3, MemR04@1
           || vtrshr || R04@1stepII2@0
45A0h TL2 max  Axh@4, Bxh@3 || max Axl@4, Bxl@3 || mov Axh@not4, MemR45@2
           || mov Axl@not4, MemR01@2 || vtrshr
           || R01@2stepII2@0, R45@2stepII2@1
D590h TL2 max  Axh@6, Bxh@5 || max Axl@6, Bxl@5 || mov Axh@not6, MemR01@2
           || mov Axl@not6, MemR45@2 || vtrshr
           || R01@2stepII2@0, R45@2stepII2@1
4A60h TL2 max  Bxh@4, Axh@3 || max Bxl@4, Axl@3 || mov Bxl@not4, MemR04@1
           || vtrshr || R04@1stepII2@0
4A64h TL2 max  Bxh@4, Axh@3 || max Bxl@4, Axl@3 || mov Bxh@not4, MemR04@1
           || vtrshr || R04@1stepII2@0
```

```
8060h TL  maxd NoReverse, Ax@8, MemR0, ge, Implied mixp, Implied r0
           || R0stepZIDS@3   ;when (r0) >= aX
8260h TL  maxd NoReverse, Ax@8, MemR0, gt, Implied mixp, Implied r0
           || R0stepZIDS@3   ;when (r0) > aX
8860h TL  min  NoReverse, Ax@8, Implied Ax@not8, Bogus MemR0, le,
           Implied mixp, Implied r0 || R0stepZIDS@3   ;when aY <= aX
8A60h TL  min  NoReverse, Ax@8, Implied Ax@not8, Bogus MemR0, lt,
           Implied mixp, Implied r0 || R0stepZIDS@3   ;when aY < aX
43C2h TL2 min  Axh@0, Axh@not0 || min Axl@0, Axl@not0 || vtrshr
D2B8h TL2 min  Axh@11, Bxh@10 || min Axl@11, Bxl@10
           || mov Axh@not11, MemR01@2 || mov Axl@not11, MemR45@2
           || vtrshr || R01@2stepII2@0, R45@2stepII2@1
4A00h TL2 min  Axh@3, Bxh@4 || min Axl@3, Bxl@4 || mov Axl@not3, MemR04@1
           || vtrshr || R04@1stepII2@0
4A04h TL2 min  Axh@3, Bxh@4 || min Axl@3, Bxl@4 || mov Axh@not3, MemR04@1
           || vtrshr || R04@1stepII2@0
45E0h TL2 min  Axh@4, Bxh@3 || min Axl@4, Bxl@3 || mov Axh@not4, MemR45@2
           || mov Axl@not4, MemR01@2 || vtrshr
           || R01@2stepII2@0, R45@2stepII2@1
D4BAh TL2 min  Axh@8, Bxh@0 || min Axl@8, Bxl@0 || vtrshr
4A20h TL2 min  Bxh@4, Axh@3 || min Bxl@4, Axl@3 || mov Bxl@not4, MemR04@1
           || vtrshr || R04@1stepII2@0
4A24h TL2 min  Bxh@4, Axh@3 || min Bxl@4, Axl@3 || mov Bxh@not4, MemR04@1
           || vtrshr || R04@1stepII2@0
47A0h TL2 mind NoReverse, Ax@3, MemR0, le, Implied mixp, Implied r0
           || R0stepZIDS@0
47A4h TL2 mind NoReverse, Ax@3, MemR0, lt, Implied mixp, Implied r0
           || R0stepZIDS@0
0080h TL  modr MemRn@0stepZIDS@3
00A0h TL  modr MemRn@0stepZIDS@3, dmod  ;Disable modulo
D294h TL2 modr MemR0123@10stepII2D2S0@0 || modr MemR4567@10stepII2D2S0@5
0D80h TL2 modr MemR0123@5stepII2D2S0@1  || modr MemR4567@5stepII2D2S0@3, dmod
0D81h TL2 modr MemR0123@5stepII2D2S0@1, dmod
           || modr MemR4567@5stepII2D2S0@3, dmod
8464h TL2 modr MemR0123@8stepII2D2S0@0, dmod || modr MemR4567@8stepII2D2S0@3
5DA0h TL2 modr MemRn@0stepD2
5DA8h TL2 modr MemRn@0stepD2, dmod
4990h TL2 modr MemRn@0stepI2
4998h TL2 modr MemRn@0stepI2, dmod
D290h TL  mov  Ab@10, Ab@5
D298h TL  mov  Abl@10, dvm
D2D8h TL  mov  Abl@10, x0
3000h TL  mov  Ablh@9, MemImm8@0
D4BCh TL  mov  Axl@8, MemImm16@16
D49Ch TL  mov  Axl@8, MemR7Imm16@16
DC80h TL  mov  Axl@8, MemR7Imm7s@0
D4B8h TL  mov  MemImm16@16, Ax@8
6100h TL  mov  MemImm8@0, Ab@11
6200h TL  mov  MemImm8@0, Ablh@10
6500h TL  mov  MemImm8@0, Axh@12, eu   ;aka Axheu
6000h TL  mov  MemImm8@0, R0123457y0@10
6D00h TL  mov  MemImm8@0, sv
D491h TL  mov  dvm, Ab@5
D492h TL  mov  icr, Ab@5
5E20h TL  mov  Imm16@16, Bx@8
5E00h TL  mov  Imm16@16, Register@0
4F80h TL  mov  Imm5u@0, icr    ;uh, but icr is 8bit wide (only 4bit are R/W)?
2500h TL  mov  Imm8s@0, Axh@12         ;signed!
2900h TL  mov  Imm8s@0, ext0
2D00h TL  mov  Imm8s@0, ext1
3900h TL  mov  Imm8s@0, ext2
3D00h TL  mov  Imm8s@0, ext3
2300h TL  mov  Imm8s@0, R0123457y0@10  ;signed!
0500h TL  mov  Imm8s@0, sv
2100h TL  mov  Imm8u@0, Axl@12         ;unsigned!
```

```
D498h TL  mov  MemR7Imm16@16, Ax@8
D880h TL  mov  MemR7Imm7s@0, Ax@8
98C0h TL  mov  MemRn@0, Bx@8 || Rn@0stepZIDS@3
1C00h TL  mov  MemRn@0, Register@5 || Rn@0stepZIDS@3
47E0h TL  mov  MemSp, Register@0
47C0h TL  mov  mixp, Register@0
2000h TL  mov  R0123457y0@9, MemImm8@0
4FC0h TL  mov  Register@0, icr
5E80h TL  mov  Register@0, mixp
1800h TL  mov  Register@5, MemRn@0 || Rn@0stepZIDS@3
5EC0h TL  mov  RegisterP0@0, Bx@5
5800h TL  mov  RegisterP0@0, Register@5
D490h TL  mov  repc, Ab@5
7D00h TL  mov  sv, MemImm8@0
D493h TL  mov  x0, Ab@5
D49Bh TL2 mov  a0h, stepi0
D59Bh TL2 mov  a0h, stepj0
4390h TL2 mov  a0h, MemR0425@2 || mov y0, MemR0425@2offsZIDZ@0
          || R0425@2stepII2D2S@0
43D0h TL2 mov  a1h, MemR0425@2 || mov y0, MemR0425@2offsZIDZ@0
          || R0425@2stepII2D2S@0
8FD4h TL2 mov  Ab@0, p0
43A0h TL2 mov  Abh@3, MemR01@2 || mov Abl@3, MemR45@2
          || R01@2stepII2@0, R45@2stepII2@1
43E0h TL2 mov  Abh@3, MemR45@2 || mov Abl@3, MemR01@2
          || R01@2stepII2@0, R45@2stepII2@1
9D40h TL2 mov  Abh@4, MemR04@1 || mov Abh@2, MemR04@1offsZI@0
          || R04@1stepII2@0
9164h TL2 mov  Abl@0, prpage
9064h TL2 mov  Abl@0, repc
D394h TL2 mov  Abl@0, x1
D384h TL2 mov  Abl@0, y1
9540h TL2 mov  Abl@3, ArArp@0
9C60h TL2 mov  Abl@3, SttMod@0
9560h TL2 mov  ArArp@0, Abl@3
D488h TL2 mov  ArArp@0, MemR04@8 || R04@8stepII2@5
5F50h TL2 mov  ArArpSttMod@0, MemR7Imm16@16
886Bh TL2 mov  Ax@8, pc
8C60h TL2 mov  Axh@4, MemR4567@8 || mov MemR0123@8, Axh@4
          || R0123@8stepII2D2S@0, R4567@8stepII2D2S@2
4800h TL2 mov  Axh@6, MemR0123@4 || movr MemR4567@4, Axh@6
          || R0123@4stepII2D2S@0, R4567@4stepII2D2S@2
4900h TL2 mov  Axh@6, MemR0123@4 || mov  MemR4567@4, Axh@6
          || R0123@4stepII2D2S@0, R4567@4stepII2D2S@2
7F80h TL2 mov  Axh@6, MemR4567@4 || movr MemR0123@4, Axh@6
          || R0123@4stepII2D2S@0, R4567@4stepII2D2S@2
8863h TL2 mov  Bx@8, pc
0008h TL2 mov  Imm16@16, ArArp@0
0023h TL2 mov  Imm16@16, r6
0001h TL2 mov  Imm16@16, repc
8971h TL2 mov  Imm16@16, stepi0
8979h TL2 mov  Imm16@16, stepj0
0030h TL2 mov  Imm16@16, SttMod@0
5DD0h TL2 mov  Imm4u@0, prpage
80C4h TL2 mov  MemR01@9, Abh@10 || mov MemR45@9, Abl@10
          || R01@9stepII2@0, R45@9stepII2@8
D292h TL2 mov  MemR0425@10_MemR0425@10offsZIDZ@5, Px@0
          || R0425@10stepII2D2S@5
D7D4h TL2 mov  MemR04@1, repc || R04@1stepII2@0
5F4Ch TL2 mov  MemR04@1, sv || sub3 MemR04@1, p0, p1, b0 || R04@1stepII2@0
D4B4h TL2 mov  MemR04@1, sv || sub3rnd MemR04@1, p0, p1, b1 || R04@1stepII2@0
DE9Ch TL2 mov  MemR04@1, sv || sub3rnd MemR04@1, p0, p1, b0 || R04@1stepII2@0
4B40h TL2 mov  MemR04@3, sv || addsub    MemR04@3, p1, p0, Bx@0
          || R04@3stepII2@2
4B42h TL2 mov  MemR04@3, sv || addsubrnd MemR04@3, p1, p0, Bx@0
```

```
          || R04@3stepII2@2
8062h TL2 mov  MemR04@4, ArArp@8  || R04@4stepII2@3
8063h TL2 mov  MemR04@4, SttMod@8 || R04@4stepII2@3
9960h TL2 mov  MemR04@4, sv || addsub    MemR04@4, p1, p0, Bx@2
          || R04@4stepD2S@3  ;<-- ordered p1, p0 here !
99E0h TL2 mov  MemR04@4, sv || addsubrnd MemR04@4, p1, p0, Bx@2
          || R04@4stepD2S@3  ;<-- ordered p1, p0 here !
9860h TL2 mov  MemR04@4, sv || sub3      MemR04@4, p0, p1, Bx@2
          || R04@4stepD2S@3
98E0h TL2 mov  MemR04@4, sv || sub3rnd   MemR04@4, p0, p1, Bx@2
          || R04@4stepD2S@3
8873h TL2 mov  MemR04@8, sv || sub3 MemR04@8, p0, p1, b1 || R04@8stepII2@3
D4C0h TL2 mov  MemR45@5, Abh@2 || mov MemR01@5, Abl@2
          || R01@5stepII2@0, R45@5stepII2@1
4D90h TL2 mov  MemR7Imm16@16, ArArpSttMod@0
D2DCh TL2 mov  MemR7Imm16@16, repc, Unused2@0, Unused1@10
1B20h TL2 mov  MemRn@0, r6 || Rn@0stepZIDS@3 ;override 1800h (mov a1,MemRn@0)
D29Ch TL2 mov  MemSp, r6, Unused2@0, Unused1@10
8A73h TL2 mov  mixp, Bx@3
4381h TL2 mov  mixp, r6
4382h TL2 mov  p0h, Bx@0
D3C2h TL2 mov  p0h, r6
4B60h TL2 mov  p0h, Register@0     ;<-- here "p0h" as source
8FD8h TL2 mov  p1, Ab@0
88D0h TL2 mov  Px@1, MemR0425@8_MemR0425@8offsZIDZ@2   || R0425@8stepII2D2S@2
88D1h TL2 mov  Px@1, MemR0425@8_MemR0425@8offsZIDZ@2,s || R0425@8stepII2D2S@2
D481h TL2 mov  r6, Bx@8
1B00h TL2 mov  r6, MemRn@0 || Rn@0stepZIDS@3 ;override 1800h (mov a0,MemRn@0)
43C1h TL2 mov  r6, mixp
5F00h TL2 mov  r6, Register@0
5F60h TL2 mov  Register@0, r6
D2D9h TL2 mov  repc, Abl@10
D7D0h TL2 mov  repc, MemR04@1 || R04@1stepII2@0
D3C8h TL2 mov  repc, MemR7Imm16@16, Unused3@0
D482h TL2 mov  stepi0, a0h
D582h TL2 mov  stepj0, a0h
D2F8h TL2 mov  SttMod@0, Abl@10
49C1h TL2 mov  x1, Ab@4
D299h TL2 mov  y1, Ab@10
5EB0h TL2 mov  prpage, Abl@0
49A0h TL2 mov  SttMod@0, MemR04@4 || R04@4stepII2@3
4DC0h TL2 mova Ab@4, MemR0425@2_MemR0425@2offsZIDZ@0 || R0425@2stepII2D2S@0
4BC0h TL2 mova MemR0425@2_MemR0425@2offsZIDZ@0, Ab@4 || R0425@2stepII2D2S@0
5F80h TL  movd MemR0123@0,ProgMemR45@2 || R0123@0stepZIDS@3, R45@2stepZIDS@5
0040h TL  movp ProgMemAxl@5, Register@0
0D40h TL2 movp ProgMemAx@5, Register@0
0600h TL  movp ProgMemRn@0, MemR0123@5 || R0123@5stepZIDS@7, Rn@0stepZIDS@3
D499h TL2 movpdw ProgMemAx@8_ProgMemAx@8offsI, pc
8864h TL  movr MemR0425@3, Abh@8 || R0425@3stepII2D2S@0   ;op*10000h+8000h
9CE0h TL  movr MemRn@0, Ax@8 || Rn@0stepZIDS@3
9CC0h TL  movr RegisterP0@0, Ax@8
5DF4h TL2 movr Bx@1, Ax@0
8961h TL2 movr r6, Ax@3
6300h TL  movs Implied sv, MemImm8@0, Ab@11
0180h TL  movs Implied sv, MemRn@0, Ab@5 || Rn@0stepZIDS@3
0100h TL  movs Implied sv, RegisterP0@0, Ab@5
5F42h TL2 movs Implied sv, r6, Ax@0
4080h TL  movsi Implied Imm5s@0, R0123457y0@9, Ab@5, Bogus Imm5s@0
D000h TL  mpy  MemR45@2, MemR0123@0 || R0123@0stepZIDS@3, R45@2stepZIDS@5
8000h TL  mpy  MemRn@0, Imm16@16    || Rn@0stepZIDS@3
8020h TL  mpy  y0, MemRn@0          || Rn@0stepZIDS@3
8040h TL  mpy  y0, Register@0
E000h TL  mpy  y0, MemImm8@0
5EA0h TL2 mpy  y0, r6
CB00h TL2 mpy  MemR45@5, MemR01@5 || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
```

```
          || sub3   p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB01h TL2 mpy  MemR45@5, MemR01@5 || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
          || sub3   p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB02h TL2 mpy  MemR45@5, MemR01@5 || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
          || sub3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB03h TL2 mpy  MemR45@5, MemR01@5 || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
          || sub3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB04h TL2 mpy  MemR45@5, MemR01@5 || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
          || add3   p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB05h TL2 mpy  MemR45@5, MemR01@5 || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
          || add3   p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB06h TL2 mpy  MemR45@5, MemR01@5 || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
          || add3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CB07h TL2 mpy  MemR45@5, MemR01@5 || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
          || add3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
D5E0h TL2 mpy  MemR04@1, x1 || mpy y1, x0 || sub3 p0, p1, Ax@3
          || R04@1stepII2@0
D5E4h TL2 mpy  MemR04@1, x1 || mpy y1, x0 || add3 p0, p1, Ax@3
          || R04@1stepII2@0
C800h TL2 mpy  MemR4567@4, MemR0123@4
          || mpy MemR4567@4offsZIDZ@2, MemR0123@4offsZIDZ@0
          || add3 p0, p1, Ab@6 || R0123@4stepII2D2S@0, R4567@4stepII2D2S@2
C900h TL2 mpy  MemR4567@4, MemR0123@4
          || mpy MemR4567@4offsZIDZ@2, MemR0123@4offsZIDZ@0
          || sub3 p0, p1, Ab@6 || R0123@4stepII2D2S@0, R4567@4stepII2D2S@2
80C2h TL2 mpy  MemR45@0, MemR01@0 || mpy MemR45@0offsZI@9, MemR01@0offsZI@8
          || add3a p0, p1, Ab@10 || R01@0stepII2@8, R45@0stepII2@9
49C8h TL2 mpy  MemR45@2, MemR01@2 || mpy MemR45@2offsZI@1, MemR01@2offsZI@0
          || sub3a p0, p1, Ab@4 || R01@2stepII2@0, R45@2stepII2@1
80C8h TL2 mpy  MemR45@2, MemR01@2 || mpy MemR45@2offsZI@1, MemR01@2offsZI@0
          || addsub  p0, p1, Ab@10 || R01@2stepII2@0, R45@2stepII2@1
81C8h TL2 mpy  MemR45@2, MemR01@2 || mpy MemR45@2offsZI@1, MemR01@2offsZI@0
          || addsuba p0, p1, Ab@10 || R01@2stepII2@0, R45@2stepII2@1
82C8h TL2 mpy  MemR45@2, MemR01@2 || mpy MemR45@2offsZI@1, MemR01@2offsZI@0
          || add     p0, p1, Ab@10 || R01@2stepII2@0, R45@2stepII2@1
83C8h TL2 mpy  MemR45@2, MemR01@2 || mpy MemR45@2offsZI@1, MemR01@2offsZI@0
          || adda    p0, p1, Ab@10 || R01@2stepII2@0, R45@2stepII2@1
00C0h TL2 mpy  MemR45@3, MemR01@3 || mpy MemR45@3offsZI@2, MemR01@3offsZI@1
          || sub  p0, p1, Ab@4 || R01@3stepII2@1, R45@3stepII2@2
00C1h TL2 mpy  MemR45@3, MemR01@3 || mpy MemR45@3offsZI@2, MemR01@3offsZI@1
          || suba p0, p1, Ab@4 || R01@3stepII2@1, R45@3stepII2@2
0D20h TL2 mpy  MemR45@3, MemR01@3 || mpyus MemR45@3offsZI@2, MemR01@3offsZI@1
          || add3a p0, p1, Ax@0, dmodi || R01@3stepII2@1, R45@3stepII2@2
0D30h TL2 mpy  MemR45@3, MemR01@3 || mpyus MemR45@3offsZI@2, MemR01@3offsZI@1
          || add3a p0, p1, Ax@0, dmodj || R01@3stepII2@1, R45@3stepII2@2
4B50h TL2 mpy  MemR45@3, MemR01@3 || mpyus MemR45@3offsZI@2, MemR01@3offsZI@1
          || add3a p0, p1, Ax@0, dmodij || R01@3stepII2@1, R45@3stepII2@2
D7A0h TL2 mpy  MemR45@3, MemR01@3 || mpy MemR45@3offsZI@2, MemR01@3offsZI@1
          || add3    sv, p0, p1, Ax@4 || R01@3stepII2@1, R45@3stepII2@2
D7A1h TL2 mpy  MemR45@3, MemR01@3 || mpy MemR45@3offsZI@2, MemR01@3offsZI@1
          || add3rnd sv, p0, p1, Ax@4 || R01@3stepII2@1, R45@3stepII2@2
9861h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3  p0, p1, Ax@8, dmodj  || R01@4stepII2@2, R45@4stepII2@3
9862h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3  p0, p1, Ax@8, dmodi  || R01@4stepII2@2, R45@4stepII2@3
9863h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3  p0, p1, Ax@8, dmodij || R01@4stepII2@2, R45@4stepII2@3
98E1h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3a p0, p1, Ax@8, dmodj  || R01@4stepII2@2, R45@4stepII2@3
98E2h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3a p0, p1, Ax@8, dmodi  || R01@4stepII2@2, R45@4stepII2@3
98E3h TL2 mpy  MemR45@4, MemR01@4 || mpy MemR45@4offsZI@3, MemR01@4offsZI@2
          || add3a p0, p1, Ax@8, dmodij || R01@4stepII2@2, R45@4stepII2@3
4DA0h TL2 mpy  y0, MemR04@3 || mpyus y1, MemR04@3offsZI@2
          || sub3  p0, p1, Ax@4 || R04@3stepII2@2
```

```
4DA1h TL2 mpy  y0, MemR04@3 || mpyus y1, MemR04@3offsZI@2
          || sub3a p0, p1, Ax@4 || R04@3stepII2@2
4DA2h TL2 mpy  y0, MemR04@3 || mpyus y1, MemR04@3offsZI@2
          || add3  p0, p1, Ax@4 || R04@3stepII2@2
4DA3h TL2 mpy  y0, MemR04@3 || mpyus y1, MemR04@3offsZI@2
          || add3a p0, p1, Ax@4 || R04@3stepII2@2
94E0h TL2 mpy  y0, MemR04@4 || mpy   y1, MemR04@4offsZI@3
          || sub3  p0, p1, Ax@8 || R04@4stepII2@3
94E2h TL2 mpy  y0, MemR04@4 || mpy   y1, MemR04@4offsZI@3
          || sub3a p0, p1, Ax@8 || R04@4stepII2@3
94E4h TL2 mpy  y0, MemR04@4 || mpy   y1, MemR04@4offsZI@3
          || add3  p0, p1, Ax@8 || R04@4stepII2@3
94E6h TL2 mpy  y0, MemR04@4 || mpy   y1, MemR04@4offsZI@3
          || add3a p0, p1, Ax@8 || R04@4stepII2@3
94E1h TL2 mpy  y0, MemR04@4 || mpysu y1, MemR04@4offsZI@3
          || sub3  p0, p1, Ax@8 || R04@4stepII2@3
94E3h TL2 mpy  y0, MemR04@4 || mpysu y1, MemR04@4offsZI@3
          || sub3a p0, p1, Ax@8 || R04@4stepII2@3
94E5h TL2 mpy  y0, MemR04@4 || mpysu y1, MemR04@4offsZI@3
          || add3  p0, p1, Ax@8 || R04@4stepII2@3
94E7h TL2 mpy  y0, MemR04@4 || mpysu y1, MemR04@4offsZI@3
          || add3a p0, p1, Ax@8 || R04@4stepII2@3
8862h TL2 mpy  y0, x1 || mpy   MemR04@4, x0 || sub3  p0, p1, Ax@8
          || R04@4stepII2@3
8A62h TL2 mpy  y0, x1 || mpy   MemR04@4, x0 || add3  p0, p1, Ax@8
          || R04@4stepII2@3
4D88h TL2 mpy  y0, x1 || mpy   y1, x0 || sub p0, p1, Ax@1
5E24h TL2 mpy  y0, x1 || mpy   y1, x0 || add p0, p1, Ab@0
8061h TL2 mpy  y0, x1 || mpy   y1, x0 || add3  p0, p1, Ab@8
8071h TL2 mpy  y0, x1 || mpy   y1, x0 || add3a p0, p1, Ab@8
8461h TL2 mpy  y0, x1 || mpy   y1, x0 || sub3  p0, p1, Ab@8
8471h TL2 mpy  y0, x1 || mpy   y1, x0 || sub3a p0, p1, Ab@8
D484h TL2 mpy  y0, x1 || mpy   y1, x0 || add3aa p0, p1, Ab@0
D49Dh TL2 mpy  y0, x1 || mpy   y1, x0 || sub p0, p1, Bx@5
D4A0h TL2 mpy  y0, x1 || mpy   y1, x0 || addsub p0, p1, Ab@0
4FA0h TL2 mpy  y0, x1 || mpy y1, x0 || add3 p0, p1, Ab@3
          || mov Axh@6, MemR04@1 || mov Bxh@2, MemR04@1offsZI@0
          || R04@1stepII2@0
5818h TL2 mpy  y0, x1 || mpy y1, x0 || addsub    sv, p0, p1, Ax@0
          || mov Axh@0, MemR0425@7 || mov Axh@not0, MemR0425@7offsZI@6
          || R0425@7stepII2@6  ;override 5800h+18h (mov a0, Register)
5838h TL2 mpy  y0, x1 || mpy y1, x0 || addsubrnd sv, p0, p1, Ax@0
          || mov Axh@0, MemR0425@7 || mov Axh@not0, MemR0425@7offsZI@6
          || R0425@7stepII2@6  ;override 5800h+38h (mov a1, Register)
80D0h TL2 mpy  y0, x1 || mpy y1, x0 || addsub    sv, p0, p1, Ax@10
          || mov Axh@9, MemR04@3 || mov Bxh@8, MemR04@3offsZI@2
          || R04@3stepII2@2
80D1h TL2 mpy  y0, x1 || mpy y1, x0 || addsubrnd sv, p0, p1, Ax@10
          || mov Axh@9, MemR04@3 || mov Bxh@8, MemR04@3offsZI@2
          || R04@3stepII2@2
80D2h TL2 mpy  y0, x1 || mpy y1, x0 || add3      sv, p0, p1, Ax@10
          || mov Axh@9, MemR04@3 || mov Bxh@8, MemR04@3offsZI@2
          || R04@3stepII2@2
80D3h TL2 mpy  y0, x1 || mpy y1, x0 || add3rnd   sv, p0, p1, Ax@10
          || mov Axh@9, MemR04@3 || mov Bxh@8, MemR04@3offsZI@2
          || R04@3stepII2@2
D3A0h TL2 mpy  y0, x1 || mpy y1, x0 || addsub p0, p1, Ab@3
          || mov Axh@6, MemR04@1 || mov Bxh@2, MemR04@1offsZI@0
          || R04@1stepII2@0
4D89h TL2 mpy  y0, x1 || mpyus y1, x0 || sub p0, p1, Ax@1
5F24h TL2 mpy  y0, x1 || mpyus y1, x0 || add p0, p1, Ab@0
8069h TL2 mpy  y0, x1 || mpyus y1, x0 || add3  p0, p1, Ab@8
8079h TL2 mpy  y0, x1 || mpyus y1, x0 || add3a p0, p1, Ab@8
8469h TL2 mpy  y0, x1 || mpyus y1, x0 || sub3  p0, p1, Ab@8
8479h TL2 mpy  y0, x1 || mpyus y1, x0 || sub3a p0, p1, Ab@8
```

```
D584h TL2 mpy  y0, x1 || mpyus y1, x0 || add3aa p0, p1, Ab@0
D59Dh TL2 mpy  y0, x1 || mpyus y1, x0 || sub p0, p1, Bx@5
D5A0h TL2 mpy  y0, x1 || mpyus y1, x0 || addsub p0, p1, Ab@0
0800h TL  mpyi NoReverse, Implied p0, y0, Imm8s@0   ;multiply  ;aka "mpys"
D100h TL  mpysu MemR45@2, MemR0123@0 || R0123@0stepZIDS@3, R45@2stepZIDS@5
8100h TL  mpysu MemRn@0, Imm16@16    || Rn@0stepZIDS@3
8120h TL  mpysu y0, MemRn@0          || Rn@0stepZIDS@3
8140h TL  mpysu y0, Register@0
CA00h TL2 mpysu MemR45@5, MemR01@5
           || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
           || sub3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA01h TL2 mpysu MemR45@5, MemR01@5
           || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
           || sub3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA02h TL2 mpysu MemR45@5, MemR01@5
           || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
           || sub3aa p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA03h TL2 mpysu MemR45@5, MemR01@5
           || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
           || sub3aa p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA04h TL2 mpysu MemR45@5, MemR01@5
           || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
           || add3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA05h TL2 mpysu MemR45@5, MemR01@5
           || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
           || add3a  p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA06h TL2 mpysu MemR45@5, MemR01@5
           || mpysu MemR45@5offsZI@4, MemR01@5offsZI@3
           || add3aa p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
CA07h TL2 mpysu MemR45@5, MemR01@5
           || mpyus MemR45@5offsZI@4, MemR01@5offsZI@3
           || add3aa p0, p1, Ab@6 || R01@5stepII2@3, R45@5stepII2@4
5EA2h TL2 mpysu y0, r6
D080h TL  msu  MemR45@2,MemR0123@0,Ax@8 || R0123@0stepZIDS@3, R45@2stepZIDS@5
90C0h TL  msu  MemRn@0, Imm16@16,  Ax@8 || Rn@0stepZIDS@3 ;multiply, subtract
9080h TL  msu  y0, MemRn@0, Ax@8 || Rn@0stepZIDS@3
90A0h TL  msu  y0, Register@0, Ax@8
B000h TL  msu  y0,MemImm8@0, Ax@8
9462h TL2 msu  y0, r6, Ax@0
8264h TL2 msusu y0, MemR0425@3, Ax@8 || R0425@3stepII2D2S@0
6790h TL  neg  Ax@12, Cond@0 ;aX=0-aX
0000h TL  nop
94C0h TL  norm Ax@8, Bogus MemRn@0 || Rn@0stepZIDS@3  ;if N=0 (aX=aX*2,rN+/-)
6780h TL  not  Ax@12, Cond@0 ;aX=not aX
D4F8h TL  or   MemImm16@16, Ax@8
A000h TL  or   MemImm8@0, Ax@8
80C0h TL  or   Imm16@16, Ax@8
C000h TL  or   Imm8u@0, Ax@8
D4D8h TL  or   MemR7Imm16@16, Ax@8
4000h TL  or   MemR7Imm7s@0, Ax@8
8080h TL  or   MemRn@0, Ax@8 || Rn@0stepZIDS@3
80A0h TL  or   RegisterP0@0, Ax@8
D291h TL2 or   Ab@10, Ax@6, Ax@5
D4A4h TL2 or   Ax@8, Bx@1, Ax@0
D3C4h TL2 or   b0, Bx@1, Ax@0
D7C4h TL2 or   b1, Bx@1, Ax@0
D388h TL2 or   r6, Ax@4
67B0h TL  pacr Implied Const8000h, Implied p0, Ax@12, Cond@0 ;aX=shfP+8000h
D7C2h TL2 pacr1 Implied Const8000h, Implied p1, Ax@0
5E60h TL  pop  Register@0
47B4h TL2 pop  Abe@0
80C7h TL2 pop  ArArpSttMod@8
0006h TL2 pop  Bx@5, Unused1@0
D7F4h TL2 pop  prpage, Unused2@0
D496h TL2 pop  Px@0
```

```
0024h TL2 pop  r6, Unused1@0
D7F0h TL2 pop  repc, Unused2@0
D494h TL2 pop  x0
D495h TL2 pop  x1
0004h TL2 pop  y1, Unused1@0
47B0h TL2 popa Ab@0
5F40h TL  push Imm16@16
5E40h TL  push Register@0
D7C8h TL2 push Abe@1, Unused1@0
D3D0h TL2 push ArArpSttMod@0
D7FCh TL2 push prpage, Unused2@0
D78Ch TL2 push Px@1, Unused1@0
D4D7h TL2 push r6, Unused1@5
D7F8h TL2 push repc, Unused2@0
D4D4h TL2 push x0, Unused1@5
D4D5h TL2 push x1, Unused1@5
D4D6h TL2 push y1, Unused1@5
4384h TL2 pusha Ax@6, Unused2@0
D788h TL2 pusha Bx@1, Unused1@0
0C00h TL  rep  Imm8u@0    ;repeat next opcode N+1 times
0D00h TL  rep  Register@0 ;repeat next opcode N+1 times
0002h TL2 rep  r6, Unused1@0
4580h TL  ret  Cond@0      ;=pop pc
D780h TL  retd    ;delayed return (after 2 clks)
45C0h TL  reti Cond@0          ;Don't context switch
45D0h TL  reti Cond@0, context ;Do context switch
D7C0h TL  retid   ;delayed, from interrupt
D3C3h TL2 retid context
0900h TL  rets Imm8u@0          ;ret+dealloc sp (for INCOMING pushed params)
67A0h TL  rnd  Implied Const8000h, Ax@12, Cond@0 ;aX=aX+8000h
6750h TL  rol  Implied Const1,     Ax@12, Cond@0 ;aX=aX rcl 1 (37bit rotate)
6F50h TL  rol  Implied Const1,     Bx@12, Cond@0 ;bX=bX rcl 1 (37bit rotate)
6740h TL  ror  Implied Const1,     Ax@12, Cond@0 ;aX=aX rcr 1 (37bit rotate)
6F40h TL  ror  Implied Const1,     Bx@12, Cond@0 ;bX=bX rcr 1 (37bit rotate)
E300h TL  rst  Imm16@16, MemImm8@0
82E0h TL  rst  Imm16@16, MemRn@0 || Rn@0stepZIDS@3
83E0h TL  rst  Imm16@16, Register@0
47B9h TL2 rst  Imm16@16, r6
4388h TL2 rst  Imm16@16, SttMod@0
E100h TL  set  Imm16@16, MemImm8@0
80E0h TL  set  Imm16@16, MemRn@0 || Rn@0stepZIDS@3
81E0h TL  set  Imm16@16, Register@0
47B8h TL2 set  Imm16@16, r6
43C8h TL2 set  Imm16@16, SttMod@0
D280h TL  shfc Implied sv, Ab@10, Ab@5, Cond@0
9240h TL  shfi Implied Imm6s@0, Ab@10, Ab@7, Bogus Imm6s@0
6720h TL  shl  Implied Const1,     Ax@12, Cond@0 ;aX=aX*2
6F20h TL  shl  Implied Const1,     Bx@12, Cond@0 ;bX=bX*2
6730h TL  shl4 Implied Const4,     Ax@12, Cond@0 ;aX=aX*10h
6F30h TL  shl4 Implied Const4,     Bx@12, Cond@0 ;bX=bX*10h
6700h TL  shr  Implied Const1,     Ax@12, Cond@0 ;aX=aX/2
6F00h TL  shr  Implied Const1,     Bx@12, Cond@0 ;bX=bX/2
6710h TL  shr4 Implied Const4,     Ax@12, Cond@0 ;aX=aX/10h
6F10h TL  shr4 Implied Const4,     Bx@12, Cond@0 ;bX=bX/10h
BA00h TL  sqr  MemImm8@0
9A80h TL  sqr  MemRn@0 || Rn@0stepZIDS@3
9AA0h TL  sqr  Register@0
D790h TL2 sqr  Abh@2 || sqr Abl@2 || add3 p0, p1, Ab@0
49C4h TL2 sqr  Abh@4 || mpysu Abh@4, Abl@4 || add3a p0, p1, Ab@0
4B00h TL2 sqr  MemR0425@4 || sqr MemR0425@4offsZIDZ@2 || add3 p0, p1, Ab@0
           || R0425@4stepII2D2S@2
5F41h TL2 sqr  r6
BC00h TL  sqra MemImm8@0, Ax@8
9C80h TL  sqra MemRn@0, Ax@8 || Rn@0stepZIDS@3
9CA0h TL  sqra Register@0, Ax@8
```

```
9062h TL2 sqra r6, Ax@8, Unused1@0
D4FFh TL  sub  MemImm16@16, Ax@8
AE00h TL  sub  MemImm8@0, Ax@8
8EC0h TL  sub  Imm16@16, Ax@8
CE00h TL  sub  Imm8u@0, Ax@8
D4DFh TL  sub  MemR7Imm16@16, Ax@8
4E00h TL  sub  MemR7Imm7s@0, Ax@8
8E80h TL  sub  MemRn@0, Ax@8 || Rn@0stepZIDS@3
8EA0h TL  sub  RegisterP0@0, Ax@8
8A61h TL2 sub  Ab@3, Bx@8
8861h TL2 sub  Bx@4, Ax@3
8064h TL2 sub  MemR01@8, sv, Abh@3 || add MemR01@8offsZI@0, sv, Abl@3
          || mov MemR45@8, sv || R01@8stepII2@0, R45@8stepII2@1
5DE0h TL2 sub  MemR04@1, sv, Abh@2 || add MemR04@1offsZI@0, sv, Abl@2
          || R04@1stepII2@0
6FC0h TL2 sub  MemR45@2, MemR01@2, Abh@3
          || add MemR45@2offsZI@1, MemR01@2offsZI@0, Abl@3
          || R01@2stepII2@0, R45@2stepII2@1
6FE0h TL2 sub  MemR45@2, MemR01@2, Abh@3
          || sub MemR45@2offsZI@1, MemR01@2offsZI@0, Abl@3
          || R01@2stepII2@0, R45@2stepII2@1
5D80h TL2 sub  MemR45@2, sv, Abh@3 || add MemR45@2offsZI@1, sv, Abl@3
          || mov MemR01@2, sv || R01@2stepII2@0, R45@2stepII2@1
5DC2h TL2 sub  p0, p1, Ab@2
D4B9h TL2 sub  p1, Ax@8
8FD0h TL2 sub  Px@1, Bx@0
D38Fh TL2 sub  r6, Ax@4
80C6h TL2 sub3 p0, p1, Ab@10
82C6h TL2 sub3a p0, p1, Ab@10
83C6h TL2 sub3aa p0, p1, Ab@10
5DC3h TL2 suba p0, p1, Ab@2
B600h TL  subh MemImm8@0, Ax@8
9680h TL  subh MemRn@0, Ax@8 || Rn@0stepZIDS@3
96A0h TL  subh Register@0, Ax@8
5E23h TL2 subh r6, Ax@8
B800h TL  subl MemImm8@0, Ax@8
9880h TL  subl MemRn@0, Ax@8 || Rn@0stepZIDS@3
98A0h TL  subl Register@0, Ax@8
5E22h TL2 subl r6, Ax@8
EF00h TL  subv Imm16@16, MemImm8@0
8EE0h TL  subv Imm16@16, MemRn@0 || Rn@0stepZIDS@3
8FE0h TL  subv Imm16@16, Register@0
47BFh TL2 subv Imm16@16, r6
4980h TL  swap SwapTypes4@0
0020h TL  trap                  ;software interrupt
A800h TL  tst0 Axl@8, MemImm8@0
8880h TL  tst0 Axl@8, MemRn@0 || Rn@0stepZIDS@3
88A0h TL  tst0 Axl@8, Register@0
E900h TL  tst0 Imm16@16, MemImm8@0
88E0h TL  tst0 Imm16@16, MemRn@0 || Rn@0stepZIDS@3
89E0h TL  tst0 Imm16@16, Register@0
D38Ch TL2 tst0 Axl@4, r6
47BCh TL2 tst0 Imm16@16, r6
9470h TL2 tst0 Imm16@16, SttMod@0
AA00h TL  tst1 Axl@8, MemImm8@0 Implied Not
8A80h TL  tst1 Axl@8, MemRn@0 Implied Not || Rn@0stepZIDS@3
8AA0h TL  tst1 Axl@8, Register@0 Implied Not
EB00h TL  tst1 Imm16@16, MemImm8@0 Implied Not
8AE0h TL  tst1 Imm16@16, MemRn@0 Implied Not || Rn@0stepZIDS@3
8BE0h TL  tst1 Imm16@16, Register@0 Implied Not
D38Dh TL2 tst1 Axl@4, r6 Implied Not
47BDh TL2 tst1 Imm16@16, r6 Implied Not
9478h TL2 tst1 Imm16@16, SttMod@0 Implied Not
80C1h TL2 tst4b a0l, MemR0425@10 || R0425@10stepII2D2S@8
4780h TL2 tst4b a0l, MemR0425@2, Ax@4 || R0425@2stepII2D2S@0
```

```
F000h TL  tstb NoReverse, Implied Not MemImm8@0, Imm4bitno@8
9020h TL  tstb NoReverse, Implied Not MemRn@0, Imm4bitno@8 || Rn@0stepZIDS@3
9000h TL  tstb NoReverse, Implied Not Register@0, Imm4bitno@8
9018h TL2 tstb NoReverse, Implied Not r6, Imm4bitno@8  ;override tstb a0,Imm4
0028h TL2 tstb NoReverse, Implied Not SttMod@0, Imm4bitno@16, Unused12@20
5F45h TL2 vtrclr vtr0         ;vtr0=0           ;for Viterbi decoding...
5F47h TL2 vtrclr vtr0, vtr1   ;vtr0=0, vtr1=0   ;(saved C/C1 carry flags)
5F46h TL2 vtrclr vtr1         ;vtr1=0
D383h TL2 vtrmov Axl@4        ;Axl=(vtr1 and FF00h)+(vtr0/100h)
D29Ah TL2 vtrmov vtr0, Axl@0  ;Axl=vtr0
D69Ah TL2 vtrmov vtr1, Axl@0  ;Axl=vtr1
D781h TL2 vtrshr              ;vtr0=vtr0/2+C*8000h, vtr1=vtr1/2+C1*8000h
D4FAh TL  xor  MemImm16@16, Ax@8
A400h TL  xor  MemImm8@0, Ax@8
84C0h TL  xor  Imm16@16, Ax@8
C400h TL  xor  Imm8u@0, Ax@8
D4DAh TL  xor  MemR7Imm16@16, Ax@8
4400h TL  xor  MemR7Imm7s@0, Ax@8
8480h TL  xor  MemRn@0, Ax@8 || Rn@0stepZIDS@3
84A0h TL  xor  RegisterP0@0, Ax@8
D38Ah TL2 xor  r6, Ax@4
8800h TL  undefined Unused5@0, Unused1@8 ;(mpy/mpys without A in bit11)
8820h TL  undefined Unused5@0, Unused1@8 ;(mpy/mpys without A in bit11)
8840h TL  undefined Unused5@0, Unused1@8 ;(mpy/mpys without A in bit11)
D800h TL  undefined Unused7@0, Unused1@8 ;(mpy/mpys without A in bit11)
9B80h TL  undefined Unused6@0  ;(sqr without A in bit8)
BB00h TL  undefined Unused8@0  ;(sqr without A in bit8)
E800h TL  undefined Unused8@0  ;(mpy without A in bit11)
5EA1h TL2 undefined Unused1@1  ;(mpy/mpys without A in bit11)
5DFCh TL2 undefined
8CDEh TL2 undefined
D3C1h TL2 undefined
5EB4h TL2 undefined Unused2@0
```

DSi TeakLite II Operand Encoding

Syntax Notes

The official Teak syntax specifies all operands in "source,dest" order, that's opposite of most other ASM languages which use "dest,source" order (except 68000 processors, which the Teak is apparently inspired on). One exception are instructions tagged "NoReverse" in the opcode list: These ones do have the operands ordered as how one would usually expect them.

Operands tagged as "Implied xx" are effectively used by the hardware, although the official syntax doesn't specify them in source code. On the other hand, operands tagged as "Bogus xx" are specified in official source code syntax, although the hardware doesn't actually use that operands in that form.

The nocash (dis-)assembler syntax reverses the operand order (except those flagged as NoReverse), removes Bogus operands, and inserts Implied operands. Moreover, immediates and memory operands are specified differently...

Memory Operands

name	native	nocash
MemRn	(Rn)	[Rn]
MemSp	(sp)	[sp]
ProgMemRn	(Rn)	[code:movpd:Rn]
ProgMemAxl	(Axl)	[code:movpd:Axl]
ProgMemAx	(Ax)	[code:Ax]
ProgMemAx_..	(Ax),(Ax+)	[code:Ax]:[code:Ax+]
MemImm8	0xNN	[page:NNh]
MemImm16	[##0xNNNN]	[NNNNh]
MemR7Imm7s	(r7+#0xNN), (r7+#-NNN)	[r7+/-NNh]
MemR7Imm16	(r7+##0xNNNN)	[r7+NNNNh]

Immediates/Addresses

```
Address18     0xNNNNN                NNNNNh         ;for bkrep/br/call
Address16     0xNNNN                 NNNNh          ;for bkrep
RelAddr7      0xNNNN                 NNNNh          ;for jmp
ImmN:         #0xNNNN                NNNNh
ImmNs:        #0xNN, #-NNN           +/-NNh
Imm16:        ##0xNNNN               NNNNh
Imm4bitno:    ...                    1 shl N
ConstZero     <implied>              0000h
Const1        <implied>              0001h
Const4        <implied>              0004h
Const8000h    <implied>              8000h
```

Operand Encoding

Below shows the binary encoding for registers/conditions. For example, "Ab@10" in the opcode list can be "b0,b1,a0,a1" encoded in bit10-11. Some instructions re-use the same bitfield for multiple operands (eg. when performing two operations on the SAME operand, or when expecting DIFFERENT operands: for "Ax@12,Ax@not12" one of the registers must "a0", and the other must be "a1").

```
Register:  RegisterP0:  Ax:     Axl:    Axh:            Px:
00: r0     00: r0       0: a0   0: a0l  0: a0h          0: p0
01: r1     01: r1       1: a1   1: a1l  1: a1h          1: p1
02: r2     02: r2
03: r3     03: r3       Bx:     Bxl:    Bxh:            Ablh:
04: r4     04: r4       0: b0   0: b0l  0: b0h          0: b0l
05: r5     05: r5       1: b1   1: b1l  1: b1h          1: b0h
06: r7     06: r7                                       2: b1l
07: y0     07: y0       Ab:     Abl:    Abh:    Abe:    3: b1h
08: st0    08: st0      0: b0   0: b0l  0: b0h  0: b0e  4: a0l
09: st1    09: st1      1: b1   1: b1l  1: b1h  1: b1e  5: a0h
0A: st2    0A: st2      2: a0   2: a0l  2: a0h  2: a0e  6: a1l
0B: p0h !! 0B: p0 !!    3: a1   3: a1l  3: a1h  3: a1e  7: a1h
0C: pc     0C: pc
0D: sp     0D: sp       Cond:
0E: cfgi   0E: cfgi     0: true  ;Always                    ;always
0F: cfgj   0F: cfgj     1: eq    ;Equal to zero             ;Z=1
10: b0h    10: b0h      2: neq   ;Not equal to zero         ;Z=0
11: b1h    11: b1h      3: gt    ;Greater than zero         ;M=0 and Z=0
12: b0l    12: b0l      4: ge    ;Greater or equal to zero  ;M=0
13: b1l    13: b1l      5: lt    ;Less than zero            ;M=1
14: ext0   14: ext0     6: le    ;Less or equal to zero     ;M=1 or Z=1
15: ext1   15: ext1     7: nn    ;Normalize flag is cleared ;N=0
16: ext2   16: ext2     8: c     ;Carry flag is set         ;C=1
17: ext3   17: ext3     9: v     ;Overflow flag is set      ;V=1
18: a0     18: a0       A: e     ;Extension flag is set     ;E=1
19: a1     19: a1       B: l     ;Limit flag is set         ;L=1
1A: a0l    1A: a0l      C: nr    ;R flag is cleared         ;R=0
1B: a1l    1B: a1l      D: niu0  ;Input user pin 0 cleared  ;IUSER0=0
1C: a0h    1C: a0h      E: iu0   ;Input user pin 0 set      ;IUSER0=1
1D: a1h    1D: a1h      F: iu1   ;Input user pin 1 set      ;IUSER1=1
1E: lc     1E: lc
1F: sv     1F: sv
```

```
R0123457y0:       Rn:                ArArpSttMod:  ArArp:       SttMod:
0: r0             0: r0              0: ar0        0: ar0       0: stt0
1: r1             1: r1              1: ar1        1: ar1       1: stt1
2: r2             2: r2              2: arp0       2: arp0      2: stt2
3: r3             3: r3              3: arp1       3: arp1      3: reserved
4: r4             4: r4              4: arp2       4: arp2      4: mod0
5: r5             5: r5              5: arp3       5: arp3      5: mod1
6: r7  ;aka rb    6: r6 ;TL2 only    6: reserved   6: reserved  6: mod2
7: y0  ;aka y     7: r7 ;TL2 only    7: reserved   7: reserved  7: mod3
```

```
                                    8: stt0
R01:     R04:     R45:              9: stt1       Ar:       BankFlags:
0: r0    0: r0    0:r4              A: stt2       0: ar0    01h: cfgi
1: r1    1: r4    1:r5              B: reserved   1: ar1    02h: r4
                                    C: mod0                 04h: r1
R0123:   R0425:   R4567:            D: mod1       Arp:      08h: r0
0: r0    0: r0    0: r4             E: mod2       0: arp0   10h: r7   ;TL2
1: r1    1: r4    1: r5             F: mod3       1: arp1   20h: cfgj ;TL2
2: r2    2: r2    2: r6                           2: arp2
3: r3    3: r5    3: r7                           3: arp3

SwapTypes:
val native           nocash         ;meaning
0:  (a0,b0)          a0,b0          ;a0 <--> b0                ;flags(a0)
1:  (a0,b1)          a0,b1          ;a0 <--> b1                ;flags(a0)
2:  (a1,b0)          a1,b0          ;a1 <--> b0                ;flags(a1)
3:  (a1,b1)          a1,b1          ;a1 <--> b1                ;flags(a1)
4:  (a0,b0),(a1,b1)  a0:a1,b0:b1    ;a0 <--> b0 and a1 <--> b1 ;flags(a0)
5:  (a0,b1),(a1,b0)  a0:a1,b1:b0    ;a0 <--> b1 and a1 <--> b0 ;flags(a0)
6:  (a0,b0,a1)       a1,b0,a0       ;a0 --> b0 --> a1          ;flags(a1)
7:  (a0,b1,a1)       a1,b1,a0       ;a0 --> b1 --> a1          ;flags(a1)
8:  (a1,b0,a0)       a0,b0,a1       ;a1 --> b0 --> a0          ;flags(a0)
9:  (a1,b1,a0)       a0,b1,a1       ;a1 --> b1 --> a0          ;flags(a0)
A:  (b0,a0,b1)       b1,a0,b0       ;b0 --> a0 --> b1          ;flags(a0)!
B:  (b0,a1,b1)       b1,a1,b0       ;b0 --> a1 --> b1          ;flags(a1)!
C:  (b1,a0,b0)       b0,a0,b1       ;b1 --> a0 --> b0          ;flags(a0)!
D:  (b1,a1,b0)       b0,a1,b1       ;b1 --> a1 --> b0          ;flags(a1)!
E:  reserved         reserved       ;-                         ;-
F:  reserved         reserved       ;-                         ;-
```

offs and step

Memory operands with "offs" allow to read from [Rn], [Rn+1], or [Rn-1]. Operands with "step" allow to increment/decrement registers (for old TL opcodes this is usually specified alongsides with memory operands, for new TL2 opcodes it's typically specified as separate "||" instruction (eg. "|| Rn@0stepZIDS@3"; which can be omitted if the step is zero).

The official syntax wants "+1,-1" abbreviated to "+,-" in some cases (but not abbreviated in other cases). step "+s" does probably refer to "stepi or stepj", but it's rather unclear which one, maybe stepi is used for r0..r3, and stepj for r4..r7, or vice versa... or maybe it depends on each opcode (particulary opcodes that allow to use "Rn" (r0..r7) might use the same step in ALL cases).

```
offsZI:                          ;maybe offsAr01 ?
 0: ''         ;Z  (zero)
 1: '+'        ;I  (increment)
offsI:
 0: '+'        ;I  (increment)
offsZIDZ:                        ;aka offsAr0123
 0: ''         ;Z  (zero)
 1: '+'        ;I  (increment)
 2: '-'        ;D  (decrement)
 3: ''         ;Z  (zero)
stepZIDS:
 0: ''         ;Z  (zero)
 1: '+1'       ;I  (increment)
 2: '-1'       ;D  (decrement)
 3: '+s'       ;S  (add step)          ;XXX ?   see "stepi" and "stepj"
modrstepZIDS:
 0: ''         ;Z  (zero)
 1: '+'        ;I  (increment)
 2: '-'        ;D  (decrement)
 3: '+s'       ;S  (add step)          ;XXX ?   see "stepi" and "stepj"
stepII2D2S:                      ;aka stepAr0123@
 0: '+1'       ;I  (increment)
 1: '+2'       ;I2 (increment twice)
 2: '-2'       ;D2 (decrement twice)
```

```
 3: '+s'       ;S  (add step)          ;XXX ?   see "stepi" and "stepj"
stepD2S:
 0: '-2'       ;D2 (decrement twice)
 1: '+s'     ' ;S  (add step)          ;XXX ?   see "stepi" and "stepj"
modrstepII2D2S0:
 0: '+'        ;I  (increment)
 1: '+2'       ;I2 (increment twice)
 2: '-2'       ;D2 (decrement twice)
 3: '+s0'      ;S0 (add step0 ?)       ;XXX ??  see "stepi0" and "stepj0"
stepII2:
 0: '+1'       ;I  (increment)
 1: '+2'       ;I2 (increment twice)
modrstepI2:
 0: '+2'       ;I2 (increment twice)
modrstepD2:
 0: '-2'       ;D2 (decrement twice)
```

Note: The "modr" opcodes are probably just incrementing/decrementing registers (with optional "modulo"), although the official syntax specifies their operands in brackets, ie. as if they were doing memory accesses.

DSi New Shared WRAM (for ARM7, ARM9, DSP)

Shared WRAM (total 800Kbytes)

```
  Old WRAM-0/1 32Kbytes (2x16K), mappable to ARM7, or ARM9
  New WRAM-A  256Kbytes (4x64K), mappable to ARM7, or ARM9
  New WRAM-B  256Kbytes (8x32K), mappable to ARM7, ARM9, or DSP-program memory
  New WRAM-C  256Kbytes (8x32K), mappable to ARM7, ARM9, or DSP-data memory
```

New WRAM mapping is done in three steps: First, releasing Slot Write Protect (on ARM7 side). Then, mapping the physical banks to logical slots (on ARM9 side). And finally, mapping those slots to actual memory addresses (on ARM7 and ARM9 sides). As an extra step, one may set Slot Write Protect flags (on ARM7 side) to prevent ARM9 from applying further changes.

```
_____________________________ Slot Write Protect _____________________________
```

MBK9 is READ/WRITE-ABLE only on ARM7 side (and READ-ONLY on ARM7).

4004060h - DSi9 - MBK9, WRAM-A/B/C Slot Write Protect (undocumented) (R)
4004060h - DSi7 - MBK9, WRAM-A/B/C Slot Write Protect (undocumented) (R/W)

```
  0-3   WRAM-A, Port 4004040h-4004043h Write (0=Writeable by ARM9, 1=Read-only)
  4-7   Unknown/Unused (0)
  8-15  WRAM-B, Port 4004044h-400404Bh Write (0=Writeable by ARM9, 1=Read-only)
  16-23 WRAM-C, Port 400404Ch-4004053h Write (0=Writeable by ARM9, 1=Read-only)
  24-31 Unknown/Unused (0)   ;but, carthdr has nonzero data for it?
```

Selects whether ARM9 may write to WRAM slot registers at 4004040h-4004053h (in Read-only mode neither ARM7 nor ARM9 can write to those registers; that applies only to that registers, ie. the memory itself isn't write-protected).

```
_____________________________ Slot Allocation _____________________________
```

MBK1-MBK5 are READ-ONLY on ARM7 side, and either READ-ONLY or READ/WRITE-ABLE on ARM9 side (depending on the MBK9 setting).

4004040h - DSi - MBK1.0, WRAM-A0 - 64K, mappable to ARM7, or ARM9
4004041h - DSi - MBK1.1, WRAM-A1 - 64K, mappable to ARM7, or ARM9
4004042h - DSi - MBK1.2, WRAM-A2 - 64K, mappable to ARM7, or ARM9
4004043h - DSi - MBK1.3, WRAM-A3 - 64K, mappable to ARM7, or ARM9

```
  0    Master (0=ARM9, 1=ARM7)
```

```
  1    Not used
  2-3  Offset (0..3) (slot 0..3) (LSB of address in 64Kbyte units)
  4-6  Not used
  7    Enable (0=Disable, 1=Enable)
```
In cooking coach, above four bytes are locked via MBK9 (not write-able, always 81h,85h,89h,8Dh)?

4004044h - DSi - MBK2.0, WRAM-B0 - 32K, mappable to ARM7, ARM9, or DSP/code
4004045h - DSi - MBK2.1, WRAM-B1 - 32K, mappable to ARM7, ARM9, or DSP/code
4004046h - DSi - MBK2.2, WRAM-B2 - 32K, mappable to ARM7, ARM9, or DSP/code
4004047h - DSi - MBK2.3, WRAM-B3 - 32K, mappable to ARM7, ARM9, or DSP/code
4004048h - DSi - MBK3.0, WRAM-B4 - 32K, mappable to ARM7, ARM9, or DSP/code
4004049h - DSi - MBK3.1, WRAM-B5 - 32K, mappable to ARM7, ARM9, or DSP/code
400404Ah - DSi - MBK3.2, WRAM-B6 - 32K, mappable to ARM7, ARM9, or DSP/code
400404Bh - DSi - MBK3.3, WRAM-B7 - 32K, mappable to ARM7, ARM9, or DSP/code
```
  0-1  Master (0=ARM9, 1=ARM7, 2 or 3=DSP/code)
  2-4  Offset (0..7) (slot 0..7) (LSB of address in 32Kbyte units)
  5-6  Not used (0)
  7    Enable (0=Disable, 1=Enable)
```

400404Ch - DSi - MBK4.0, WRAM-C0 - 32K, mappable to ARM7, ARM9, or DSP/data
400404Dh - DSi - MBK4.1, WRAM-C1 - 32K, mappable to ARM7, ARM9, or DSP/data
400404Eh - DSi - MBK4.2, WRAM-C2 - 32K, mappable to ARM7, ARM9, or DSP/data
400404Fh - DSi - MBK4.3, WRAM-C3 - 32K, mappable to ARM7, ARM9, or DSP/data
4004050h - DSi - MBK5.0, WRAM-C4 - 32K, mappable to ARM7, ARM9, or DSP/data
4004051h - DSi - MBK5.1, WRAM-C5 - 32K, mappable to ARM7, ARM9, or DSP/data
4004052h - DSi - MBK5.2, WRAM-C6 - 32K, mappable to ARM7, ARM9, or DSP/data
4004053h - DSi - MBK5.3, WRAM-C7 - 32K, mappable to ARM7, ARM9, or DSP/data
```
  0-1  Master (0=ARM9, 1=ARM7, 2 or 3=DSP/data)
  2-4  Offset (0..7) (slot 0..7) (LSB of address in 32Kbyte units)
  5-6  Not used (0)
  7    Enable (0=Disable, 1=Enable)
```

Address Mapping

MBK6-8 exist as separate READ/WRITE-ABLE registers on ARM7 and ARM9 side (making it six registers in total).

4004054h - DSi - MBK6, WRAM-A, 64K..256K mapping (R/W)
```
  0-3   Not used (0)
  4-11  Start Address (3000000h+N*10000h)     ;=3000000h..3FF0000h
  12-13 Image Size (0 or 1=64KB/Slot0, 2=128KB/Slot0+1+2??, 3=256KB/Slot0..3)
  14-19 Not used (0)
  20-28 End Address   (3000000h+N*10000h-1)   ;=2FFFFFFh..4FEFFFFh
  29-31 Not used (0)
```

4004058h - DSi - MBK7, WRAM-B (R/W)
400405Ch - DSi - MBK8, WRAM-C (R/W)
```
  0-2   Not used (0)
  3-11  Start Address (3000000h+N*8000h)      ;=3000000h..3FF8000h
  12-13 Image Size (0=32K/Slot0,1=64KB/Slot0-1,2=128KB/Slot0-3,3=256KB/Slot0-7)
  14-18 Not used (0)
  19-28 End Address   (3000000h+N*8000h-1)    ;=2FFFFFFh..4FF7FFFh
  29-31 Not used (0)
```

Uh, but, ARM7 3800000h..3FFFFFFh contains OTHER memory (ARM7-WRAM) !?

Notes

Slots and Image Size vs Start/End Addresses
When using Image Size of 4 slots, then Memory at 3000000h..3FFFFFFh is:
```
Slots  0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,etc.
```
When start=6, and End=12, then (with above example), only following is mapped:
```
Slots  -,-,-,-,-,-,2,3,0,1,2,3,-,-,-,-,etc.
```
Observe that the mapped region starts with Slot 2 (not Slot 0) in that case.
Moreover, some slots may be empty (disabled, or mapped to another CPU), so, if Slot 3 is empty (disabled or mapped to another CPU), then memory appears to look somewhat as so:
```
Slots  -,-,-,-,-,-,2,z,0,1,2,z,-,-,-,-,etc.
```
Whereas, the "z" areas seem to read as zerofilled memory blocks (rather than mirroring to underlaying WRAM's of lower priority).

Overlapping WRAM regions
```
New Shared-WRAM-A   Highest Priority
New Shared-WRAM-B   High Priority
New Shared-WRAM-C   Low Priority
Old Shared-WRAM-0/1 Lowest Priority
Old ARM7-WRAM       Whatever Priority (unknown...)
I/O ports 4xxxxxxh  Whatever Priority (unknown...)
```
Overlapping WRAM slots
```
Unknown what happens when selecting multiple WRAM blocks to the same slot?
```

The initial MBK values are derived from carthdr.
Exploits for DSi cartridges & DSiware usually have ARM7.MBK registers disabled via SCFG_EXT7, making it impossible to change that ARM7 registers; the ARM9.MBK registers are usually kept enabled, nethertheless, the ARM7.MBK9 setting can apply some restrictions to ARM9 side (for example, in Cooking Coach, WRAM-A is controlled via ARM7.MBK1, so ARM9 can control WRAM-B and WRAM-C via ARM9.MBK2-5 only).

DSi New DMA (NDMA)

The DSi has four New DMA channels for ARM7 and ARM9 each (eight New DMA channels in total). The old NDS-style DMA channels do still exist, too [though unknown which priority they have in relation to new channels].

4004100h - DSi - NDMAGCNT NewDMA Global Control (R/W) [00000000h]
```
0-15  Unused (0)
16-19 Cycle Selection (0=None, 1..15=1..16384 clks) ;1 SHL (N-1)
20-30 Unused (0)
31    DMA Arbitration Mode (0=NDMA0=HighestPriority, 1=RoundRobinPriority)
```
CycleSelection is used ONLY in RoundRobin mode; if so... then it does specify the number of cycles that can be executed by ARM9 and DSP <CPUs?> during NDMA?

4004104h+x*1Ch - NDMAxSAD - NewDMAx Source Address (R/W) [00000000h]
4004108h+x*1Ch - NDMAxDAD - NewDMAx Destination Address (R/W) [00000000h]
```
0-1   Unused (0)
2-31  DMA Source/Destination Address, in 4-byte steps
```

400410Ch+x*1Ch - NDMAxTCNT - NewDMAx Total Length for Repeats (R/W) [0]
```
0-27  Total Number of Words to Transfer (1..0FFFFFFFh, or 0=10000000h)
28-31 Unused (0)
```
Not used in "Start immediately" mode (which doesn't repeat).
Not used in "Repeat infinitely" mode (which repeats forever).
Used only in "Repeat until NDMAxTCNT" mode (for example, to define the total size of the Camera picture).
Total Length isn't required to be a multiple of the Logical or Physical Block Sizes (for example the DSi launcher

uses Total=64h with Log=8/Phys=8; in that case only 4 words (instead of 8 words) are transferred for the last block).

4004110h+x*1Ch - NDMAxWCNT - NewDMAx Logical Block Size

```
  0-23  Number of Words to Transfer       (1..00FFFFFFh, or 0=01000000h)
  24-31 Unused (0)
```

Should be a multiple of the Physical Block Size specified in NDMAxCNT.Bit16-19.
The bus will be monopolized until the selected number of words for (physical) block transfers has completed, a single (physical) block transfer cycle will never be split up.

4004114h+x*1Ch - NDMAxBCNT - NewDMAx Block Transfer Timing/Interval

```
  0-15  Interval Timer (1..FFFFh, or 0=Infinite/TillTransferEnd)
  16-17 Prescaler (33.514MHz SHR (n*2)) ;0=33MHz, 1=8MHz, 2=2MHz, 3=0.5MHz
  18-31 Unused (0)
```

Allows to insert a delay after each (Physical?) Block.

4004118h+x*1Ch - NDMAxFDATA - NewDMAx Fill Data

```
  0-31  Fill Data (can be used as Fixed Source Data for memfill's)
```

This value is used when setting NDMAxCNT.Bit13-14=3, which causes the source data to be read directly (within 0 clock cycles) from the NDMAxFDATA (instead of from the address specified in NDMAxSAD; in this case, the NDMAxSAD is ignored/don't care).

400411Ch+x*1Ch - NDMAxCNT - NewDMAx Control

```
  0-9   Unused (0)
  10-11 Dest Address Update   (0=Increment, 1=Decrement, 2=Fixed, 3=Reserved)
  12    Dest Address Reload   (0=No, 1=Reload at (logical blk?) transfer end)
  13-14 Source Address Update (0=Increment, 1=Decrement, 2=Fixed, 3=FillData)
  15    Source Address Reload (0=No, 1=Reload at (logical blk?) transfer end)
  16-19 Physical Block Size   (0..0Fh=1..32768 words, aka (1 SHL n) words)
  20-23 Unused (0)
  24-28 DMA Startup Mode      (00h..1Fh, see ARM7/ARM9 startup lists below)
  29    DMA Repeat Mode       (0=Repeat until NDMAxTCNT, 1=Repeat infinitely)
  30    DMA Interrupt Enable  (0=Disable, 1=Enable)
  31    DMA Enable/Busy       (0=Disable, 1=Enable/Busy)
```

Startup Modes for ARM9:

```
  00h      Timer0                       ;\
  01h      Timer1                       ; new NDMA-specific modes
  02h      Timer2                       ;
  03h      Timer3                       ;/
  04h      DS Cartridge Slot
  05h      Reserved (maybe 2nd DS-Cart Slot, or GBA slot relict?)
  06h      V-Blank
  07h      H-Blank (but not during V-blank)
  08h      Display Sync (sync to H-blank drawing) ;Uh, what is BLANK-DRAWING ??
  09h      Work RAM (what?) (=probably Main memory display, as on NDS)
  0Ah      Geometry Command FIFO
  0Bh      Camera                       ;-new NDMA-specific mode
  0Ch..0Fh Reserved
  10h..1Fh Start immediately (without repeat)
```

Startup Modes for ARM7:

```
  00h      Timer0                       ;\
  01h      Timer1                       ; new NDMA-specific modes
  02h      Timer2                       ;
  03h      Timer3                       ;/
  04h      DS Cartridge Slot
  05h      Reserved? (maybe 2nd DS-Cart Slot, or GBA slot relict?)
  06h      V-Blank
  07h      NDS-Wifi
  08h      SD/MMC   (SD_DATA32_FIFO)    ;\
```

```
09h      DSi-Wifi (SDIO_DATA32_FIFO)  ;
0Ah      AES in   (AES_WRFIFO)        ; new NDMA-specific modes
0Bh      AES out  (AES_RDFIFO)        ;
0Ch      Microphone (MIC_DATA)        ;/
0Dh..0Fh Reserved?
10h..1Fh Start immediately (without repeat)
```

Start/repeat modes
There are three different transfer modes.
1) Start immediately (without repeat):
the transfer ends after one Logical Block, without repeat. With single IRQ (after last/only block).
2) Start by Hardware events, Repeat until NDMAxTCNT:
the transfer repeats Logical Blocks until reaching the Total Length. With single IRQ (after last block).
3) Start by Hardware events, Repeat infinitely:
the transfer repeats Logical Blocks infinitely. With multiple IRQs (one IRQ after EACH logical block).

Read-only Effect
There is something that can make port 4004104h..4004173h read-only. For example, when FFh-filling all DSi registers, and then 00h-filling them, then most DMA bits stay set (00h-filling them another time does clear them).
Maybe, during enabled transfers, ONLY the enable/busy bit is writeable?

DSi Microphone and SoundExt

4004600h - DSi7 - MIC_CNT - Microphone Control (can be 0000E10Fh)

```
0-1   Data Format    (0=MakeStereo, 1=SameAsNormal?, 2=Normal, 3=None)  (R/W)
2-3   Sampling Rate  (0..3=F/1, F/2, F/3, F/4)                          (R/W)
4-7   Unused (0)                                                          (-)
8     FIFO Empty     (0=No, 1=Empty)           ;0 words                  (R)
9     FIFO Half-Full (0=No, 1=Half-Full)       ;8 or more words          (R)
10    FIFO Full      (0=No, 1=Full)            ;16 words                 (R)
11    FIFO Overrun   (0=No, 1=Overrun/Stopped) ;more than 16 words       (R)
12    Clear FIFO     (0=No change, 1=Clear)    ;works only if bit15 was 0 (W)
13-14 IRQ Enable     (0=Off, 1=Same as 3, 2=When Full, 3=When Half-Full)(R/W)
15    Enable         (0=Disable, 1=Enable)                               (R/W)
```

The Sampling Rate depends on the I2S frequency in SNDEXCNT.Bit13,

```
I2S=32.73kHz --> F/1=32.73kHz, F/2=16.36kHz, F/3=10.91kHz, F/4=8.18kHz
I2S=47.61kHz --> F/1=47.61kHz, F/2=23.81kHz, F/3=15.87kHz, F/4=11.90kHz
```

The Sampling Rate becomes zero (no data arriving) when SNDEXCNT.Bit15=0, or when MIC_CNT.bit0-1=3, or when MIC_CNT.bit15=0, or when Overrun has occurred.

4004604h - DSi7 - MIC_DATA - Microphone Data (R)
The internal (and external) microphones are mono only. However, the "MakeStereo" data format can convert "mono to stereo" (ie. each sample is output twice, and 1st/2nd sample contain the same value; accordingly, the FIFO is getting full twice as fast).
MIC_DATA can be read by software, or via NDMA (with startup mode 0Ch and Blocksize 8 words).

```
0-15  Signed 16bit Data, 1st sample    ;\16 words FIFO, aka 32 halfwords
16-31 Signed 16bit Data, 2nd sample    ;/
```

Overrun Caution: If data isn't read fast enough then the Overrun flag will get set, and FIFO updates will be stopped. At that point one can still read the remaining 16 words from the FIFO, but no further words will be added to the FIFO (until clearing the FIFO and Overrun flag via MIC_CNT.bit12; which requires bit15=0 before setting bit12).

Microphone Touchscreen/sound control
DSi Touchscreen/Sound Init Flowcharts
The Sample Data tends to be 0000h or FFFFh if the microphone related TSC-registers aren't unmuted.

4004700h - DSi7 - SNDEXCNT (R/W)

```
  0-3     NITRO/DSP ratio                   (valid range is 0 to 8)       (R/W)
  4-12    Unknown/Unused (0)                                               (0?)
  13      Sound/Microphone I2S frequency (0=32.73 kHz, 1=47.61 kHz)  (R or R/W)
  14      Mute status (does NOT affect mic)          (?=Mute WHAT?)       (R/W)
  15      Enable Microphone (and Sound Output!)          (1=Enable)       (R/W)
```

Bit13 is write-able only if bit15 was zero before the write.

NITRO/DSP ratio

The DSP can generate sound output aswell, alongside the old NITRO sound mixer. The following settings configure the ratio between DSP and NITRO mixer output:

```
  00h      DSP sound 8/8, NITRO sound 0/8 (=DSP sound only)
  01h      DSP sound 7/8, NITRO sound 1/8
  02h      DSP sound 6/8, NITRO sound 2/8
  03h      DSP sound 5/8, NITRO sound 3/8
  04h      DSP sound 4/8, NITRO sound 4/8 (=half volume for DSP and NITRO each)
  05h      DSP sound 3/8, NITRO sound 5/8
  06h      DSP sound 2/8, NITRO sound 6/8
  07h      DSP sound 1/8, NITRO sound 7/8
  08h      DSP sound 0/8, NITRO sound 8/8 (=NITRO sound only)
  09h..0Fh Reserved
```

Uh, what is that? Hopefully, a volume-ratio? Preferably, no time-ratio!

DSi Advanced Encryption Standard (AES)

AES I/O Ports

DSi AES I/O Ports

AES Pseudo Code

Little Endian Code (as used in DSi hardware):

DSi AES Little-Endian High Level Functions

DSi AES Little-Endian Core Function and Key Schedule

DSi AES Little-Endian Tables and Test Values

Big Endian Code (as used more commonly, in non-DSi implementations):

DSi AES Big-Endian High Level Functions

DSi AES Big-Endian Core Function and Key Schedule

DSi AES Big-Endian Tables and Test Values

Most AES values are endian-free byte-strings, so different "endianness" does just mean to reverse the byte order of the 16/24/32-byte KEYs, the 16-byte data chunks, and the 16-byte CTR/CFB/CBC/MAC registers (in some of the latter cases it's also referring to actual endiannes, eg. for CTR increments).

AES Usage in DSi

AES-CCM is used for several SD/MMC files (using a custom Nintendo-specific CCM variant; consisting of 128K-byte data blocks with 32-byte footers):

DSi ES Block Encryption

AES-CTR is used for the Modcrypt areas defined in Cartridge Header, and for eMMC Boot Sectors and for eMMC MBR/Partitions.

AES Usage in DSi-Wifi

DSi Wifi is also supporting AES (and TKIP and WEP) encryption, the Wifi unit contains additional AES hardware for data packet encryption (however, Wifi EAPOL handshakes do require AES key unwrap implemented in software).

AES Usage in DSi-Shop

DSi Shop downloads (and system updates) are using big-endian AES-CBC, this does require an AES software

implementation because the DSi's AES hardware couldn't decrypt that AES variant.

DSi AES I/O Ports

4004400h - DSi7 - AES_CNT (parts R/W)

```
  0-4   Write FIFO Count    (00h..10h words) (00h=Empty, 10h=Full)          (R)
  5-9   Read FIFO Count     (00h..10h words) (00h=Empty, 10h=Full)          (R)
  10    Write FIFO Flush    (0=No change, 1=Flush)                    (N/A or W)
  11    Read FIFO Flush     (0=No change, 1=Flush)                    (N/A or W)
  12-13 Write FIFO DMA Size (0..3 = 16,12,8,4 words) (2=Normal=8)     (R or R/W)
  14-15 Read FIFO DMA Size  (0..3 = 4,8,12,16 words) (1=Normal=8)     (R or R/W)
  16-18 CCM MAC Size, max(4,(N*2+2)) bytes, usually 7=16 bytes        (R or R/W)
  19    CCM Pass Associated Data to RDFIFO (0=No/Normal, 1=Yes)       (R or R/W)
          Bit19=1 is a bit glitchy: The data should theoretically arrive in
          RDFIFO immediately after writing 4 words to WRFIFO, but actually,
          Bit19=1 seems to cause 4 words held hidden in neither FIFO, until
          the first Payload block is written (at that point, the hidden
          associated words are suddenly appearing in RDFIFO)
  20    CCM MAC Verify Source (0=From AES_WRFIFO, 1=From AES_MAC)     (R or R/W)
  21    CCM MAC Verify Result (0=Invalid/Busy, 1=Verified/Okay)             (R)
  22-23 Unknown/Unused (0)                                                  (0)
  24    Key Select        (0=No change, 1=Apply key selected in Bit26-27)   (W)
  25    Key Schedule Busy (uh, always 0=ready?) (rather sth else busy?)     (R)
  26-27 Key Slot          (0..3=KEY0..KEY3, applied via Bit24)        (R or R/W)
  28-29 Mode (0=CCM/decrypt, 1=CCM/encrypt, 2=CTR, 3=Same as 2)       (R or R/W)
  30    Interrupt Enable  (0=Disable, 1=Enable IRQ on Transfer End)   (R or R/W)
  31    Start/Enable      (0=Disable/Ready, 1=Enable/Busy)                (R/W)
```

Bit31 gets cleared automatically shortly after all data (as indicated in AES_BLKCNT) is written to WRFIFO, and the IRQ is generated alongsides; the transfer isn't fully completed at that point since there may be still data (and CCM/encrypt MAC result) in RDFIFO.

4004404h - DSi7 - AES_BLKCNT (W)

Specifies the transfer length, counted in 16-byte blocks.

```
  0-15  Number of Extra associated data blocks for AES-CCM (unused for AES-CTR)
  16-31 Number of Payload data blocks (0..FFFFh = 0..FFFF0h bytes)
```

The length values are copied to internal counter registers on transfer start (the value in AES_BLKCNT is left unchanged during/after transfer).

4004408h - DSi7 - AES_WRFIFO (W)
400440Ch - DSi7 - AES_RDFIFO (R)

```
  0-31  Data
```

Writing to WRFIFO works even when AES_CNT.bit31=0 (the data does then stay in WRFIFO though, and doesn't arrive in RDFIFO).

4004420h - DSi7 - AES_IV (16 bytes) (W)

This contains the Initialization Vector (aka IV aka Nonce). The hardware does use that value to automatically initialize the internal CTR/CBC registers when starting encryption/decryption:

```
  For AES-CTR mode:  CTR[00h..0Fh] = AES_IV[00h..0Fh]
                     CBC[00h..0Fh] = not used by AES-CTR mode
  For AES-CCM mode:  CTR[00h..0Fh] = 00h,00h,00h,AES_IV[00h..0Bh],02h
                     CBC[00h..0Fh] = x0h,xxh,0xh,AES_IV[00h..0Bh],flg
```

The initial CTR/CBC values for AES-CCM mode are following the CCM specifications, but WITHOUT encoding the "extra associated data size" in upper bytes of first block (see CCM pseudo code chapter for details). The CTR/CBC registers are manipulated during transfer, however, the AES_IV content is kept unchanged during/after transfer.

4004430h - DSi7 - AES_MAC (16 bytes) (W)

The MAC (Message Authentication Code) is an encrypted checksum, computed alongsides with the actual data encryption/decryption, and used only in AES-CCM mode. There are three ways how the DSi deals with MAC values:

```
AES-CCM Encryption: MAC is returned in AES_RDFIFO after transfer
AES-CCM Decryption, AES_CNT.20=0: MAC written to AES_WRFIFO after transfer
AES-CCM Decryption, AES_CNT.20=1: MAC written to AES_MAC before transfer
```

The AES_MAC register and the RDFIFO/WRFIFO blocks are always 16-byte wide; when selecting a smaller MAC size in AES_CNT, then the lower bytes of that 16-byte value are 00h-padded (eg. a 6-byte MAC would appear as 00000000h, 00000000h, xxxx0000h, xxxxxxxxh), for ENCRYPT those 00h-bytes are returned in RDFIFO, for DECRYPT those padding bytes MUST be 00h (else the verification will fail).
The minimum MAC size is 4 bytes (trying to use 2 byte by setting AES_CNT.16-18 to 00h is producing the exact same result as when setting it to 01h, ie. 4-bytes)

4004440h - DSi7 - AES_KEY0 (48 bytes) (W)
4004470h - DSi7 - AES_KEY1 (48 bytes) (W)
40044A0h - DSi7 - AES_KEY2 (48 bytes) (W)
40044D0h - DSi7 - AES_KEY3 (48 bytes) (W)

```
Byte 00h-0Fh  Normal 128bit Key      ;\use either normal key,
Byte 10h-1Fh  Special 128bit Key_X   ; or special key_x/y
Byte 20h-2Fh  Special 128bit Key_Y   ;/
```

Writing the last word of "Key_Y" (or any of its last four bytes, ie. byte(s) 2Ch..2Fh) causes the Normal Key to be overwritten by following values:

```
Key = ((Key_X XOR Key_Y) + FFFEFB4E295902582A680F5F1A4F3E79h) ROL 42
```

After changing a key, one must (re-)apply it via AES_CNT.Bits 24,26-27.

DMA
The AES data would be usually transferred via two NDMA channels, one for WRFIFO, one for RDFIFO. The NDMAs should be started BEFORE setting AES_CNT.31 (else the DMA will miss the first WRFIFO data request; and DMA won't start). The DMAs 'Logical Block' sizes should match up with the block sizes selected in AES_CNT (a bigger logical block size would cause FIFO overruns/underruns, a smaller logical block size could work theoretically, but it practice it causes the DMA to hang after the first data request; apparently data requests are somewhat generated upon "empty-not-empty" transitions, rather than upon "enough data/space" status).

Reading Write-Only Values
The AES_IV register and the AES_KEY registers are fully write-able, including 8bit STRB writes; this allows to 'read' the write-only values via brute-force without any noticeable delay (ie. encrypt 16 bytes with original values, then change one byte to values 00h..FFh, and check which of those values gives same encryption result). AES_BLKCNT can be also dumped by simple counting.

Cartheader Key Request Byte
The firmware is usually destroying the AES_KEY registers before starting DSi programs. However, bits in CartHeader[1B4h] allow to "request" certain keys to be left intact.

DSi BIOS & Firmware Keys
The DSi BIOS contains several AES keys in the non-dumpable upper 32K halves; most of that keys are relocated to RAM/TCM, so they can be dumped via main memory hacks (there might be some further keys that cannot be dumped, in case they exist only in early boot stages).

DSi AES Little-Endian High Level Functions

AES-CTR (Counter)
aes_crypt_ctr(src,dst,len,nc_off,iv)

```
aes_setkey(ENCRYPT,key,key_size]                    ;-init key
[ctr+0..15] = [iv+0..15]                            ;-init ctr
```

```
  n=[nc_off]
  while len>0   ;code is 100% same for ENCRYPT and DECRYPT         ;\
    if n=0                                                        ; encrypt
      aes_crypt_block(ENCRYPT,ctr,tmp)                            ; or decrypt
      littleendian(ctr)=littleendian(ctr)+1   ;increment counter  ; message
    [dst] = [src] xor [tmp+n]                                     ;
    src=src+1, dst=dst+1, len=len-1, n=(n+1) and 0Fh              ;/
  [nc_off]=n
```

AES-CCM (Counter with CBC-MAC)

aes_ccm_crypt(mode,src,dst,msg_len,iv,iv_len,xtra,xtra_len,mac,mac_len)

```
  if mac_len<4 or mac_len>16 or (mac_len and 1)=1 then error       ;\limits
  if iv_len<7 or iv_len>13 then error                              ;/
  aes_setkey(ENCRYPT,key,key_size]                                 ;-init key
  ctr_len = 15-iv_len                                              ;\
  [ctr+15]=ctr_len-1          ;bit3..7=zero   ;1 byte (ctr_len)    ; init ctr
  [ctr+(15-iv_len)..14] = [iv+0..(iv_len-1)]  ;7..13 bytes (iv)    ;
  [ctr+0..(14-iv_len)]=littleendian(0)  ;8..2 bytes (counter=0)    ;/
  [cbc+0..15]=littleendian(msg_len)  ;-[(iv_len+1)..15]=msg_len    ;\
  if [cbc+15..15-iv_len]<>0 then error  ;msg_len overlaps iv/flags ;
  [cbc+(15-iv_len)..14]=[iv+0..iv_len-1] ;-[1..iv_len]=iv/nonce    ;
  [cbc+15].bit7=0  ;reserved/zero   ;\                             ; init cbc
  [cbc+15].bit6=(xtra_len>0)        ; [15]=flags                   ;
  [cbc+15].bit5..3=(mac_len/2-1)    ;                              ;
  [cbc+15].bit2..0=(ctr_len-1)      ;/                             ;
  aes_crypt_block(ENCRYPT,cbc,cbc)      ;UPDATE_CBC_MAC            ;/
  if NintendoDSi then                                              ;\
    a=0 ;the DSi hardware doesn't support xtra_len encoding at all ;
  elseif xtra_len<0FF00h then                                      ;
    [cbc+14..15]=[cbc+14..15] xor littleendian(xtra_len), a=2      ; weird
  elseif xtra_len<100000000h then                                  ; encoding
    [cbc+14..15]=[cbc+14..15] xor littleendian(FFFEh)              ; for
    [cbc+10..13]=[cbc+10..13] xor littleendian(xtra_len), a=6      ; xtra_len
  else                                                             ;
    [cbc+14..15]=[cbc+14..15] xor littleendian(FFFFh)              ;
    [cbc+6..13] =[cbc+6..13]  xor littleendian(xtra_len), a=10     ;/
  while xtra_len>0                                                 ;\scatter
    z=min(xtra_len,16-a)                                           ; cbc by
    [cbc+16-a-z..(15-a)]=[cbc+16-a-z..(15-a)] xor [xtra+0..(z-1)]  ; xtra
    aes_crypt_block(ENCRYPT,cbc,cbc)    ;UPDATE_CBC_MAC            ; (if any)
    xtra=xtra+z, xtra_len=xtra_len-z, a=0                          ;/
  while msg_len>0                                                  ;\
    littleendian(ctr)=littleendian(ctr)+1   ;increment counter     ;
    aes_crypt_block(ENCRYPT,ctr,tmp)    ;CTR_CRYPT                 ;
    z=min(msg_len,16)                                              ; encrypt
    if mode=ENCRYPT                                                ; or decrypt
      [cbc+(16-z)..15] = [cbc+(16-z)..15] xor [src+0..(z-1)]       ; message
    [dst+0..(z-1)] = [src+0..(z-1)] xor [tmp+(16-z)..15]           ; body
    if mode=DECRYPT                                                ;
      [cbc+(16-z)..15] = [cbc+(16-z)..15] xor [dst+0..(z-1)]       ;
    aes_crypt_block(ENCRYPT,cbc,cbc)    ;UPDATE_CBC_MAC            ;
    src=src+z, dst=dst+z, msg_len=msg_len-z                        ;/
  [ctr+0..(14-iv_len)]=littleendian(0)  ;reset counter=0           ;\
  aes_crypt_block(ENCRYPT,ctr,tmp)      ;CTR_CRYPT                 ; message
  [cbc+0..15] = [cbc+0..15] xor [tmp+0..15]                        ; auth code
  z=mac_len                                                        ; (mac)
  IF mode=ENCRYPT then [mac+0..(z-1)] = [cbc+(16-z)..15]           ;
  IF mode=DECRYPT and [mac+0..(z-1)] <> [cbc+(16-z)..15] then error;/
```

AES-Key-Wrap/Unwrap

This is used for EAPOL Key Data in WPA2 Wifi packets, however, neither DSi-ARM7 nor DSi-Wifi do support that by hardware, so the unwrap (decrypt) must be implemented by software. For details see:
[DS Wifi WPA/WPA2 Encryption]

Below are some other AES variants (just for curiosity - those variants aren't used in DSi):

AES-CBC (Cipher-block chaining)

aes_crypt_cbc(mode,src,dst,len,iv)

```
  aes_setkey(mode,key,key_size]                                    ;-init key
  [cbc+0..15] = [iv+0..15]                                         ;-init cbc
  if (len AND 0Fh)>0 then error
  while len>0                                                      ;\
    if mode=ENCRYPT                                                ;
      [dst+0..15] = [src+0..15] xor [cbc+0..15]                    ;
      aes_crypt_block(mode,dst,dst)                                ; encrypt
      [cbc+0..15] = [dst+0..15]                                    ; or decrypt
    if mode=DECRYPT                                                ; message
      [tmp+0..15] = [src+0..15]                                    ;
      aes_crypt_block(mode,src,dst)                                ;
      [dst+0..15] = [dst+0..15] xor [cbc+0..15]                    ;
      [cbc+0..15] = [tmp+0..15]                                    ;
    src=src+16, dst=dst+16, len=len-16                             ;/
```

AES-CFB128 (Cipher feedback on 128bits, aka 16 bytes)

aes_crypt_cfb128(mode,src,dst,len,iv_off,iv)

```
  aes_setkey(ENCRYPT,key,key_size]                                 ;-init key
  [cfb+0..15] = [iv+0..15]                                         ;-init cfb
  n=[iv_off]
  while len>0                                                      ;\
    if n=0 then aes_crypt_block(ENCRYPT,cfb,cfb)                   ; encrypt
    if mode=DECRYPT then c=[src], [dst]=c xor [cfb+n], [cfb+n]=c   ; or decrypt
    if mode=ENCRYPT then c=[cfb+n] xor [src], [cfb+n]=c, [dst]=c   ; message
    src=src+1, dst=dst+1, len=len-1, n=(n+1) and 0Fh               ;/
  [iv_off]=n
```

AES-CFB8 (Cipher feedback on 8bits, aka 1 byte, very inefficient)

aes_crypt_cfb8(mode,src,dst,len,iv)

```
  aes_setkey(ENCRYPT,key,key_size]                                 ;-init key
  [cfb+0..15] = [iv+0..15]                                         ;-init cfb
  n=[iv_off]
  while len>0                                                      ;\
    aes_crypt_block(ENCRYPT,cfb,tmp)                               ;
    [cfb+1..15] = [cfb+0..14]   ;shift with 8-bit step             ; encrypt
    if mode=DECRYPT then [cfb+0] = [src+(n xor 0Fh)]               ; or decrypt
    [dst+(n xor 0Fh)] = [src+(n xor 0Fh)] xor [tmp+15]  ;shift-in  ; message
    if mode=ENCRYPT then [cfb+0] = [dst+(n xor 0Fh)]               ;
    len=len-1, n=n+1                                               ;/
  [iv_off]=n
```

AES-ECB (Electronic codebook, very basic, very insecure)

aes_crypt_ecb(mode,src,dst,len)

```
  aes_setkey(mode,key,key_size]                                    ;-init key
  if (len AND 0Fh)>0 then error
  while len>0                                                      ;\encrypt
    aes_crypt_block(mode,src,dst)                                  ; or decrypt
    src=src+16, dst=dst+16, len=len-16                             ;/message
```

DSi AES Little-Endian Core Function and Key Schedule

```
 aes_crypt_block(mode,src,dst):
  Y0 = RK[0] xor [src+00h]
  Y1 = RK[1] xor [src+04h]
  Y2 = RK[2] xor [src+08h]
```

```
 Y3 = RK[3] xor [src+0Ch]
 ;below code depending on mode:      <---ENCRYPT--->  -or-  <---DECRYPT--->
 for i=1 to nr-1
   X0      = RK[i*4+0] xor scatter32(FT,Y1,Y2,Y3,Y0)  -or-  (RT,Y3,Y2,Y1,Y0)
   X1      = RK[i*4+1] xor scatter32(FT,Y2,Y3,Y0,Y1)  -or-  (RT,Y0,Y3,Y2,Y1)
   X2      = RK[i*4+2] xor scatter32(FT,Y3,Y0,Y1,Y2)  -or-  (RT,Y1,Y0,Y3,Y2)
   X3      = RK[i*4+3] xor scatter32(FT,Y0,Y1,Y2,Y3)  -or-  (RT,Y2,Y1,Y0,Y3)
   Y0=X0, Y1=X1, Y2=X2, Y3=X3
 [dst+00h] = RK[nr*4+0] xor scatter8(FSb,Y1,Y2,Y3,Y0) -or-  (RSb,Y3,Y2,Y1,Y0)
 [dst+04h] = RK[nr*4+1] xor scatter8(FSb,Y2,Y3,Y0,Y1) -or-  (RSb,Y0,Y3,Y2,Y1)
 [dst+08h] = RK[nr*4+2] xor scatter8(FSb,Y3,Y0,Y1,Y2) -or-  (RSb,Y1,Y0,Y3,Y2)
 [dst+0Ch] = RK[nr*4+3] xor scatter8(FSb,Y0,Y1,Y2,Y3) -or-  (RSb,Y2,Y1,Y0,Y3)

scatter32(TAB,a,b,c,d):              scatter8(TAB,a,b,c,d):
 w=      (TAB[a.bit0..7] ror 24)      w.bit0..7   = TAB[a.bit0..7]
 w=w xor (TAB[b.bit8..15] ror 16)     w.bit8..15  = TAB[b.bit8..15]
 w=w xor (TAB[c.bit16..23] ror 8)     w.bit16..23 = TAB[c.bit16..23]
 w=w xor (TAB[d.bit24..31])           w.bit24..31 = TAB[d.bit24..31]
 return w                             return w

aes_setkey(mode,key,keysize):  ;out: RK[0..43/51/59], nr=10/12/14
 aes_generate_tables   ;<-- unless tables are already initialized
 if keysize<>128 and keysize<>192 and keysize<>256 then error  ;size in bits
 rc=01h, j=0, jj=keysize/32, nr=jj+6   ;jj=4,6,8      ;\
 for i=0 to (nr+1)*4-1                 ;nr=10,12,14   ; copy 16/24/32-byte key
   if i<jj then w=[key+(jj-1-i)*4+0..3]               ; to RK[0..3/5/7]
   else w=w xor RK[(i-jj) xor 3]                      ; and, make
   RK[i xor 3]=w, j=j+1                               ; RK[4/6/8..43/51/59]
   if j=jj then                                       ;
     w=scatter8(FSb,w,w,w,w)                          ;
     w=(w rol 8) xor (rc shl 24)                      ;
     j=0, rc=rc*2, if rc>0FFh then rc=rc xor 11Bh     ;
   if j=4 and jj=8 then w=scatter8(FSb,w,w,w,w)       ;/
 if mode=DECRYPT then
   for i=0 to nr/2-1     ;swap entries (except middle one)
     for j=0 to 3
       w=RK[i*4+j], v=RK[nr*4-i*4+j]
       RK[i*4+j]=v, RK[nr*4-i*4+j]=w
   for i=4 to nr*4-1     ;modify entries (except RK[0..3] and RK[nr*4+0..3])
     w=RK[i], w=scatter8(FSb,w,w,w,w), RK[i]=scatter32(RT,w,w,w,w)
```

DSi AES Little-Endian Tables and Test Values

```
aes_generate_tables:
 for i=0 to 0FFh               ;compute pow and log tables...
   if i=0 then x=01h, else x=x xor x*2, if x>0FFh then x=x xor 11Bh
   pow[i]=x, log[x]=i
 for i=0 to 0FFh               ;generate the forward and reverse S-boxes...
   x=pow[0FFh-log[i]]
   x=x xor (x rol 1) xor (x rol 2) xor (x rol 3) xor (x rol 4) xor 63h
   if i=0 then x=63h
   FSb[i]=x, RSb[x]=i
 for i=0 to 0FFh               ;generate the forward and reverse tables...
   x=FSb[i]*2, if x>0FFh then x=x xor 11Bh
   FT[i]=(FSb[i]*00010101h) xor (x*01000001h)
   w=00000000h, x=RSb[i]
   if x<>00h then   ;ie. not at i=63h
     w=w+pow[(log[x]+log[0Eh]) mod 00FFh]*1000000h
     w=w+pow[(log[x]+log[09h]) mod 00FFh]*10000h
     w=w+pow[(log[x]+log[0Dh]) mod 00FFh]*100h
     w=w+pow[(log[x]+log[0Bh]) mod 00FFh]*1h
   RT[i]=w
```

```
aes_generate_tables_results:
 pow[00h..FFh] = 01,03,05,0F,11,..,C7,52,F6,01   ;pow  ;\needed temporarily
 log[00h..FFh] = 00,FF,19,01,32,..,C0,F7,70,07   ;log  ;/for table creation
 FSb[00h..FFh] = 63,7C,77,7B,F2,..,B0,54,BB,16   ;Forward S-box
 RSb[00h..FFh] = 52,09,6A,D5,30,..,55,21,0C,7D   ;Reverse S-box
 FT[00h..FFh] = C66363A5,F87C7C84,..,2C16163A    ;Forward Table
 RT[00h..FFh] = 51F4A750,7E416553,..,D0B85742    ;Reverse Table

aes_setkey_results:
 key = "AES-Test-Key-Str-1234567-Abcdefg"  ;use only 1st bytes for 128/192bit
 128bit ENCRYPT --> RK[0..9..30..43] = 2D534541..2783080F..93AF7DF0..827EE10D
 192bit ENCRYPT --> RK[0..9..30..51] = 79654B2D..9708FA95..2529372B..C66C19FA
 256bit ENCRYPT --> RK[0..9..30..59] = 3332312D..DF5C92A5..74174E2E..3C8ADAE6
 128bit DECRYPT --> RK[0..9..30..43] = AEABCD4D..ECD33F19..8C87B246..7274532D
 192bit DECRYPT --> RK[0..9..30..51] = AFA9796F..72A3EFE5..455646C7..37363534
 256bit DECRYPT --> RK[0..9..30..59] = 0ED52830..4601F929..415A7D65..67666564

aes_crypt_results:
 [key+0..15]    = "AES-Test-Key-Str-1234567-Abcdefg"
 [iv+0..15]     = "Nonce/InitVector"
 [xtra+0..20]   = "Extra-Associated-Data"  ;\for CCM
 iv_len=12, mac_len=16, xtra_len=xx        ;/
 Unencrypted:   [dta+0..113Fh] = "Unencrypted-Data", 190h x "TestPadding"
 AES-ECB:       [dta+0..113Fh] = 20,24,73,88,..,44,A8,D6,A8  ;\
 AES-CBC:       [dta+0..113Fh] = A4,6F,7A,F2,..,58,C9,02,B4  ;
 AES-CFB128:    [dta+0..113Fh] = 20,C6,DB,35,..,9A,83,7F,DB  ; keysize=128
 AES-CFB8:      [dta+0..113Fh] = 55,C7,75,1C,..,24,6E,A6,D1  ;
 AES-CTR:       [dta+0..113Fh] = 20,C6,DB,35,..,AB,09,0C,75  ;
 AES-CCM:       [dta+0..113Fh] = C8,37,D7,F1,..,7B,EF,FC,12  ;
 AES-CCM (ori): [mac+0..0Fh]   = xx,xx,xx,xx,..,xx,xx,xx,xx  ;
 AES-CCM (DSi): [mac+0..0Fh]   = xx,xx,xx,xx,..,xx,xx,xx,xx  ;/
 AES-ECB:       [dta+0..113Fh] = CC,B6,4D,17,..,D3,56,3E,64  ;-keysize=192
 AES-ECB:       [dta+0..113Fh] = A9,A9,9B,3E,..,8A,C6,13,A1  ;-keysize=256
```

DSi AES Big-Endian High Level Functions

AES-CTR (Counter)

aes_crypt_ctr(src,dst,len,nc_off,iv)

```
  aes_setkey(ENCRYPT,key,key_size]                                  ;-init key
  [ctr+0..15] = [iv+0..15]                                          ;-init ctr
  n=[nc_off]
  while len>0   ;code is 100% same for ENCRYPT and DECRYPT          ;\
    if n=0                                                          ; encrypt
      aes_crypt_block(ENCRYPT,ctr,tmp)                              ; or decrypt
      bigendian(ctr)=bigendian(ctr)+1     ;increment counter        ; message
    [dst] = [src] xor [tmp+n]                                       ;
    src=src+1, dst=dst+1, len=len-1, n=(n+1) and 0Fh                ;/
  [nc_off]=n
```

AES-CCM (Counter with CBC-MAC)

aes_ccm_crypt(mode,src,dst,msg_len,iv,iv_len,xtra,xtra_len,mac,mac_len)

```
  if mac_len<4 or mac_len>16 or (mac_len and 1)=1 then error        ;\limits
  if iv_len<7 or iv_len>13 then error                               ;/
  aes_setkey(ENCRYPT,key,key_size]                                  ;-init key
  ctr_len = 15-iv_len                                               ;\
  [ctr+0]=ctr_len-1     ;bit3..7=zero   ;1 byte (ctr_len)           ; init ctr
  [ctr+1..iv_len] = [iv+0..(iv_len-1)]  ;7..13 bytes (iv)           ;
  [ctr+(iv_len+1)..15]=bigendian(0)     ;8..2 bytes (counter=0)     ;/
  [cbc+0..15]=bigendian(msg_len)   ;-[(iv_len+1)..15]=msg_len       ;\
  if [cbc+0..iv_len]<>0 then error ;errif msg_len overlaps iv/flags;
```

```
[cbc+1..iv_len]=[iv+0..iv_len-1] ;-[1..iv_len]=iv (aka nonce)    ;
[cbc+0].bit7=0  ;reserved/zero   ;\                              ; init cbc
[cbc+0].bit6=(xtra_len>0)        ; [0]=flags                     ;
[cbc+0].bit5..3=(mac_len/2-1)    ;                               ;
[cbc+0].bit2..0=(ctr_len-1)      ;/                              ;
aes_crypt_block(ENCRYPT,cbc,cbc)      ;UPDATE_CBC_MAC            ;/
if NintendoDSi then                                              ;\
  a=0 ;the DSi hardware doesn't support xtra_len encoding at all ;
elseif xtra_len<0FF00h then                                      ;
  [cbc+0..1]=[cbc+0..1] xor bigendian(xtra_len), a=2             ; weird
elseif xtra_len<100000000h then                                  ; encoding
  [cbc+0..1]=[cbc+0..1] xor bigendian(FFFEh)                     ; for
  [cbc+2..5]=[cbc+2..5] xor bigendian(xtra_len), a=6             ; xtra_len
else                                                             ;
  [cbc+0..1]=[cbc+0..1] xor bigendian(FFFFh)                     ;
  [cbc+2..9]=[cbc+2..9] xor bigendian(xtra_len), a=10            ;/
while xtra_len>0                                                 ;\scatter
  z=min(xtra_len,16-a)                                           ; cbc by
  [cbc+a..(a+z-1)]=[cbc+a..(a+z-1)] xor [xtra+0..(z-1)]          ; xtra
  aes_crypt_block(ENCRYPT,cbc,cbc)    ;UPDATE_CBC_MAC            ; (if any)
  xtra=xtra+z, xtra_len=xtra_len-z, a=0                          ;/
while msg_len>0                                                  ;\
  bigendian(ctr)=bigendian(ctr)+1     ;increment counter         ;
  aes_crypt_block(ENCRYPT,ctr,tmp)    ;CTR_CRYPT                 ;
  z=min(msg_len,16)                                              ; encrypt
  if mode=ENCRYPT                                                ; or decrypt
    [cbc+0..(z-1)] = [cbc+0..(z-1)] xor [src+0..(z-1)]           ; message
  [dst+0..(z-1)] = [src+0..(z-1)] xor [tmp+0..(z-1)]             ; body
  if mode=DECRYPT                                                ;
    [cbc+0..(z-1)] = [cbc+0..(z-1)] xor [dst+0..(z-1)]           ;
  aes_crypt_block(ENCRYPT,cbc,cbc)    ;UPDATE_CBC_MAC            ;
  src=src+z, dst=dst+z, msg_len=msg_len-z                        ;/
[ctr+(iv_len+1)..15]=bigendian(0)     ;reset counter=0           ;\
aes_crypt_block(ENCRYPT,ctr,tmp)      ;CTR_CRYPT                 ; message
[cbc+0..15] = [cbc+0..15] xor [tmp+0..15]                        ; auth code
z=mac_len                                                        ; (mac)
IF mode=ENCRYPT then [mac+0..(z-1)] = [cbc+0..(z-1)]             ;
IF mode=DECRYPT and [mac+0..(z-1)] <> [cbc+0..(z-1)] then error  ;/
```

AES-Key-Wrap/Unwrap
This is used for EAPOL Key Data in WPA2 Wifi packets, however, neither DSi-ARM7 nor DSi-Wifi do support that by hardware, so the unwrap (decrypt) must be implemented by software. For details see:
DS Wifi WPA/WPA2 Encryption

Below are some other AES variants (just for curiosity - those variants aren't used in DSi):

AES-CBC (Cipher-block chaining)
aes_crypt_cbc(mode,src,dst,len,iv)

```
  aes_setkey(mode,key,key_size]                                  ;-init key
  [cbc+0..15] = [iv+0..15]                                       ;-init cbc
  if (len AND 0Fh)>0 then error
  while len>0                                                    ;\
    if mode=ENCRYPT                                              ;
      [dst+0..15] = [src+0..15] xor [cbc+0..15]                  ;
      aes_crypt_block(mode,dst,dst)                              ; encrypt
      [cbc+0..15] = [dst+0..15]                                  ; or decrypt
    if mode=DECRYPT                                              ; message
      [tmp+0..15] = [src+0..15]                                  ;
      aes_crypt_block(mode,src,dst)                              ;
      [dst+0..15] = [dst+0..15] xor [cbc+0..15]                  ;
      [cbc+0..15] = [tmp+0..15]                                  ;
    src=src+16, dst=dst+16, len=len-16                           ;/
```

AES-CFB128 (Cipher feedback on 128bits, aka 16 bytes)

aes_crypt_cfb128(mode,src,dst,len,iv_off,iv)

```
  aes_setkey(ENCRYPT,key,key_size]                                  ;-init key
  [cfb+0..15] = [iv+0..15]                                          ;-init cfb
  n=[iv_off]
  while len>0                                                       ;\
    if n=0 then aes_crypt_block(ENCRYPT,cfb,cfb)                    ; encrypt
    if mode=DECRYPT then c=[src], [dst]=c xor [cfb+n], [cfb+n]=c    ; or decrypt
    if mode=ENCRYPT then c=[cfb+n] xor [src], [cfb+n]=c, [dst]=c    ; message
    src=src+1, dst=dst+1, len=len-1, n=(n+1) and 0Fh                ;/
  [iv_off]=n
```

AES-CFB8 (Cipher feedback on 8bits, aka 1 byte, very inefficient)

aes_crypt_cfb8(mode,src,dst,len,iv)

```
  aes_setkey(ENCRYPT,key,key_size]                                  ;-init key
  [cfb+0..15] = [iv+0..15]                                          ;-init cfb
  while len>0                                                       ;\
    aes_crypt_block(ENCRYPT,cfb,tmp)                                ;
    [cfb+0..14] = [cfb+1..15]   ;shift with 8-bit step              ; encrypt
    if mode=DECRYPT then [cfb+15] = [src]                           ; or decrypt
    [dst] = [src] xor [tmp+0]   ;shift-in new 8-bits                ; message
    if mode=ENCRYPT then [cfb+15] = [dst]                           ;
    src=src+1, dst=dst+1, len=len-1                                 ;/
```

AES-ECB (Electronic codebook, very basic, very insecure)

aes_crypt_ecb(mode,src,dst,len)

```
  aes_setkey(mode,key,key_size]                                     ;-init key
  if (len AND 0Fh)>0 then error
  while len>0                                                       ;\encrypt
    aes_crypt_block(mode,src,dst)                                   ; or decrypt
    src=src+16, dst=dst+16, len=len-16                              ;/message
```

DSi AES Big-Endian Core Function and Key Schedule

```
 aes_crypt_block(mode,src,dst):
  Y0 = RK[0] xor [src+00h]
  Y1 = RK[1] xor [src+04h]
  Y2 = RK[2] xor [src+08h]
  Y3 = RK[3] xor [src+0Ch]
  ;below code depending on mode:      <---ENCRYPT--->  -or-  <---DECRYPT--->
  for i=1 to nr-1
    X0      = RK[i*4+0] xor scatter32(FT,Y0,Y1,Y2,Y3)  -or-  (RT,Y0,Y3,Y2,Y1)
    X1      = RK[i*4+1] xor scatter32(FT,Y1,Y2,Y3,Y0)  -or-  (RT,Y1,Y0,Y3,Y2)
    X2      = RK[i*4+2] xor scatter32(FT,Y2,Y3,Y0,Y1)  -or-  (RT,Y2,Y1,Y0,Y3)
    X3      = RK[i*4+3] xor scatter32(FT,Y3,Y0,Y1,Y2)  -or-  (RT,Y3,Y2,Y1,Y0)
    Y0=X0, Y1=X1, Y2=X2, Y3=X3
  [dst+00h] = RK[nr*4+0] xor scatter8(FSb,Y0,Y1,Y2,Y3) -or-  (RSb,Y0,Y3,Y2,Y1)
  [dst+04h] = RK[nr*4+1] xor scatter8(FSb,Y1,Y2,Y3,Y0) -or-  (RSb,Y1,Y0,Y3,Y2)
  [dst+08h] = RK[nr*4+2] xor scatter8(FSb,Y2,Y3,Y0,Y1) -or-  (RSb,Y2,Y1,Y0,Y3)
  [dst+0Ch] = RK[nr*4+3] xor scatter8(FSb,Y3,Y0,Y1,Y2) -or-  (RSb,Y3,Y2,Y1,Y0)

 scatter32(TAB,a,b,c,d):              scatter8(TAB,a,b,c,d):
  w=      (TAB[a.bit0..7])             w.bit0..7   = TAB[a.bit0..7]
  w=w xor (TAB[b.bit8..15] rol 8)      w.bit8..15  = TAB[b.bit8..15]
  w=w xor (TAB[c.bit16..23] rol 16)    w.bit16..23 = TAB[c.bit16..23]
  w=w xor (TAB[d.bit24..31] rol 24)    w.bit24..31 = TAB[d.bit24..31]
  return w                             return w

 aes_setkey(mode,key,keysize):  ;out: RK[0..43/51/59], nr=10/12/14
```

```
 aes_generate_tables   ;<-- unless tables are already initialized
 if keysize<>128 and keysize<>192 and keysize<>256 then error  ;size in bits
 rc=01h, j=0, jj=keysize/32, nr=jj+6   ;jj=4,6,8      ;\
 for i=0 to (nr+1)*4-1                 ;nr=10,12,14   ; copy 16/24/32-byte key
   if i<jj then w=[key+i*4+0..3]                      ; to RK[0..3/5/7]
   else w=w xor RK[i-jj]                              ; and, make
   RK[i]=w, j=j+1                                     ; RK[4/6/8..43/51/59]
   if j=jj then                                       ;
     w=scatter8(FSb,w,w,w,w)                          ;
     w=(w ror 8) xor (rc)                             ;
     j=0, rc=rc*2, if rc>0FFh then rc=rc xor 11Bh     ;
   if j=4 and jj=8 then w=scatter8(FSb,w,w,w,w)       ;/
 if mode=DECRYPT then
   for i=0 to nr/2-1     ;swap entries (except middle one)
     for j=0 to 3
       w=RK[i*4+j], v=RK[nr*4-i*4+j]
       RK[i*4+j]=v, RK[nr*4-i*4+j]=w
   for i=4 to nr*4-1     ;modify entries (except RK[0..3] and RK[nr*4+0..3])
     w=RK[i], w=scatter8(FSb,w,w,w,w), RK[i]=scatter32(RT,w,w,w,w)
```

DSi AES Big-Endian Tables and Test Values

aes_generate_tables:

```
 for i=0 to 0FFh               ;compute pow and log tables...
   if i=0 then x=01h, else x=x xor x*2, if x>0FFh then x=x xor 11Bh
   pow[i]=x, log[x]=i
 for i=0 to 0FFh               ;generate the forward and reverse S-boxes...
   x=pow[0FFh-log[i]]
   x=x xor (x rol 1) xor (x rol 2) xor (x rol 3) xor (x rol 4) xor 63h
   if i=0 then x=63h
   FSb[i]=x, RSb[x]=i
 for i=0 to 0FFh               ;generate the forward and reverse tables...
   x=FSb[i]*2, if x>0FFh then x=x xor 11Bh
   FT[i]=(FSb[i]*01010100h) xor (x*01000001h)
   w=00000000h, x=RSb[i]
   if x<>00h then   ;ie. not at i=63h
     w=w+pow[(log[x]+log[0Eh]) mod 00FFh]*1h
     w=w+pow[(log[x]+log[09h]) mod 00FFh]*100h
     w=w+pow[(log[x]+log[0Dh]) mod 00FFh]*10000h
     w=w+pow[(log[x]+log[0Bh]) mod 00FFh]*1000000h
   RT[i]=w
```

aes_generate_tables_results:

```
 pow[00h..FFh] = 01,03,05,0F,11,..,C7,52,F6,01   ;pow  ;\needed temporarily
 log[00h..FFh] = 00,FF,19,01,32,..,C0,F7,70,07   ;log  ;/for table creation
 FSb[00h..FFh] = 63,7C,77,7B,F2,..,B0,54,BB,16   ;Forward S-box
 RSb[00h..FFh] = 52,09,6A,D5,30,..,55,21,0C,7D   ;Reverse S-box
 FT[00h..FFh] = A56363C6,847C7CF8,..,3A16162C    ;Forward Table
 RT[00h..FFh] = 50A7F451,5365417E,..,4257B8D0    ;Reverse Table
```

aes_setkey_results:

```
 key = "AES-Test-Key-Str-1234567-Abcdefg"  ;use only 1st bytes for 128/192bit
 128bit ENCRYPT --> RK[0..9..30..43] = 2D534541..ED0DC6FA..43DAC81C..0F5026BB
 192bit ENCRYPT --> RK[0..9..30..51] = 2D534541..4AAB3D82..29CA38D2..CA4DFE3B
 256bit ENCRYPT --> RK[0..9..30..59] = 2D534541..1AA51359..CCB886C8..88956C9C
 128bit DECRYPT --> RK[0..9..30..43] = F653079B..47DD8A1C..1C2070A7..7274532D
 192bit DECRYPT --> RK[0..9..30..51] = 3CEC6AFF..C4F96B6F..AE36B4AE..7274532D
 256bit DECRYPT --> RK[0..9..30..59] = DE7ADCD9..8C559ADD..067A387E..7274532D
```

aes_crypt_results:

```
 [key+0..15]    = "AES-Test-Key-Str-1234567-Abcdefg"
 [iv+0..15]     = "Nonce/InitVector"
```

```
[xtra+0..20]   = "Extra-Associated-Data"  ;\for CCM
iv_len=12, mac_len=16, xtra_len=21        ;/
Unencrypted:   [dta+0..113Fh] = "Unencrypted-Data", 190h x "TestPadding"
AES-ECB:       [dta+0..113Fh] = 5F,BD,04,DB,..,E4,07,F4,B6  ;\
AES-CBC:       [dta+0..113Fh] = 0B,BB,53,FA,..,DD,28,6D,AE  ;
AES-CFB128:    [dta+0..113Fh] = F4,75,4F,0E,..,73,B5,D7,E7  ; keysize=128
AES-CFB8:      [dta+0..113Fh] = F4,10,6A,83,..,BF,1B,16,3E  ;
AES-CTR:       [dta+0..113Fh] = F4,75,4F,0E,..,04,DF,EB,BA  ;
AES-CCM:       [dta+0..113Fh] = FD,1A,6D,98,..,EE,FD,68,F6  ;
AES-CCM (ori): [mac+0..0Fh]   = FD,F9,FE,85,..,4F,50,3C,AF  ;
AES-CCM (DSi): [mac+0..0Fh]   = xx,xx,xx,xx,..,xx,xx,xx,xx  ;/
AES-ECB:       [dta+0..113Fh] = 0E,69,F5,1A,..,9A,5F,7A,9A  ;-keysize=192
AES-ECB:       [dta+0..113Fh] = C6,FB,68,C1,..,14,89,6C,E0  ;-keysize=256
```

DSi ES Block Encryption

ES Block Encryption, for lack of a better name, is a Nintendo DSi specific data encryption method. It's used for some SD/MMC files:

```
FAT16:\sys\dev.kp
FAT16:\ticket\000300tt\4gggggggg.tik (tickets)
SD Card: .bin files (aka Tad Files)
twl-*.der files (within the "verdata" NARC file)
```

Block Layout

```
00000h      BLKLEN   Data Block      (AES-CCM encrypted)
BLKLEN+00h  10h      Data Checksum   (AES-CCM MAC value on above Data)
BLKLEN+10h  1        Fixed 3Ah       (AES-CTR encrypted)
BLKLEN+11h  0Ch      Nonce           (unencrypted)
BLKLEN+1Dh  1        BLKLEN.bit16-23 (AES-CTR encrypted)
BLKLEN+1Eh  1        BLKLEN.bit8-15  (AES-CTR encrypted)
BLKLEN+1Fh  1        BLKLEN.bit0-7   (AES-CTR encrypted)
```

BLKLEN can be max 20000h. If the Data is bigger than 128Kbytes, then it's split into multiple block(s) with BLKLEN=20000h (the last block can have smaller BLKLEN).

Data Block Encryption/Decryption (AES-CCM)

```
IV[00h..0Bh]=[BLKLEN+11h..1Ch]  ;Nonce
IV[0Ch..0Fh]=Don't care (not used for CCM)
```

With that IV value, apply AES-CCM on the Data Block:

```
00000h      BLKLEN   Data Block      (AES-CCM)
```

Observe that some DSi files have odd BLKLEN values, so you may need to append padding bytes to the Data Block (the DSi hardware requires full 16-byte chunks for encryption/decryption).

Data Block Padding (16-byte alignment)

For encryption, it's simple: Just append 00h byte(s) as padding value.

For decryption, it's more complicated: The padding values should be ENCRYPTED 00h-bytes (required to get the same MAC result as for encryption). If you don't want to verify the MAC, then you could append whatever dummy bytes. If you want to verify the MAC, then you could pre-calculate the padding values as so:

```
IV[00h..02h]=BLKLEN/10h + 1     ;CTR value for last 16-byte block
IV[03h..0Eh]=[BLKLEN+11h..1Ch]  ;Nonce
IV[0Fh]=02h                     ;Indicate 3-byte wide CTR (fixed on DSi)
```

Then, use AES-CTR (not CCM) to encrypt sixteen 00h-bytes, the last byte(s) of the result can be then used as padding value(s). The padding values should be pre-calculated BEFORE starting the CCM decryption (the DSi hardware allows only one AES task at once, so they cannot be calculated via AES-CTR when AES-CCM decryption is in progress).

Verifying the Footer values (AES-CTR)

This step is needed only for verification purposes (encryption tools should create these values, but decryption tools may or may not verify them).

```
IV[00h]=00h                    ;Zero
IV[01h..0Ch]=[BLKLEN+11h..1Ch] ;Nonce
IV[0Dh..0Fh]=00h,00h,00h       ;Zero
```

With that IV value (and same Key as for AES-CCM), apply AES-CTR on the last 16 bytes of the block:

```
BLKLEN+10h  1        Fixed 3Ah       (AES-CTR encrypted)
BLKLEN+11h  0Ch      Nonce           (unencrypted)
BLKLEN+1Dh  1        BLKLEN.bit16-23 (AES-CTR encrypted)
BLKLEN+1Eh  1        BLKLEN.bit8-15  (AES-CTR encrypted)
BLKLEN+1Fh  1        BLKLEN.bit0-7   (AES-CTR encrypted)
```

AES-CTR is XORing the data stream (encrypted bytes will turn into unencrypted bytes, and vice-versa), so the result would look as so:

```
BLKLEN+10h  1        Fixed 3Ah       (unencrypted)       (to be verified)
BLKLEN+11h  0Ch      Nonce           (AES-CTR encrypted) (useless/garbage)
BLKLEN+1Dh  1        BLKLEN.bit16-23 (unencrypted)       (to be verified)
BLKLEN+1Eh  1        BLKLEN.bit8-15  (unencrypted)       (to be verified)
BLKLEN+1Fh  1        BLKLEN.bit0-7   (unencrypted)       (to be verified)
```

Mind that BLKLEN can be odd, so data at BLKLEN+00h..1Fh isn't necessarily located at 4-byte aligned addresses.

DSi Cartridge Header

Old NDS Header Entries

The first 180h bytes of the DSi Header are essentially same as on NDS:

DS Cartridge Header

New/changed entries in DSi carts are:

```
012h 1    Unitcode (00h=NDS, 02h=NDS+DSi, 03h=DSi) (bit1=DSi)
01Ch 1    NDS: Reserved / DSi: Flags (03h=Normal, 0Bh=Sys, 0Fh=Debug/Sys)
            bit0 Has TWL-Exclusive Region      ;MUST be 1 for DSi titles?
            bit1 Modcrypted (0=No, 1=Yes, see [220h..22Fh])
            bit2 Modcrypt key select (0=Retail, 1=Debug)
            bit3 Disable Debug ?
01Dh 1    NDS: Region   / DSi: Permit jump (00h=Normal, 01h=System Settings)
                          bit0 jump (always include title at [2FFD800h..])
                          bit1 tmpjump (?)
068h 4    Icon/Title offset (same as NDS, but with new extra entries)
080h 4    Total Used ROM size, EXCLUDING DSi area
088h 4    NDS: Reserved / ARM9 Parameters Table Offset ???  ;base=[028h]
08Ch 4    NDS: Reserved / ARM7 Parameters Table Offset ???  ;base=[038h]
090h 2    NDS: Reserved / NTR ROM Region End/80000h   ;\usually both same
092h 2    NDS: Reserved / TWL ROM Region Start/80000h ;/(zero for DSiware)
```

NDS carts (that don't use DSi features, but are manufactured after DSi release) contains these extra entries:

```
(012h)1   Unitcode (must be 00h for non-DSi carts)
(020h)16  Changed ARM9/ARM7 areas (DSi-in-NDS-mode more restricted than NDS)
088h 4    Unknown (B8h,4Bh,00h,00h) (similar as in DSi carts)
1BFh 1    Flags (40h=RSA+TwoHMACs, 60h=RSA+ThreeHMACs)
33Ch 14   HMAC for Icon/Title (only if [1BFh]=60h)      ;as Whitelist Phase 3
378h 14   HMAC for 160h-byte header and ARM9+ARM7 areas ;as Whitelist Phase 1
38Ch 14   HMAC for OverlayARM9+NitroFAT (zero if no overlay)      ;as Phase 2
F80h 128  RSA signature
```

All other entries at 160h..FFFh are zerofilled in non-DSi carts.

Changed ARM9/ARM7 areas (and new ARM9i/ARM7i areas)

NDS allowed ARM9 or ARM7 to occupy about 3.8MB of main RAM. On DSi this is restriced to 2.5MB for ARM9 and 0.25MB for ARM7, ie. 2.75MB in total, this restriction applies even in NDS-mode, thus making DSi not fully backwards compatible with NDS games (officially licensed NDS titles hopefully don't conflict with that new restriction). In DSi mode, one can additionally load 2.5MB for ARM9i and 1MB for ARM7i, ie.

6.25MB in total for all four areas.
The DSi loading is a bit complicated, the areas are first loaded to DEDICATED areas:

```
ARM9  2004000h..227FFFFh (siz=27C000h) (for NDS mode: 2000000h and up)
ARM7  2380000h..23BFFFFh (siz=40000h)
ARM9i 2400000h..267FFFFh (siz=280000h)
ARM7i 2E80000h..2F87FFFh (siz=108000h)
```

If the cart header does match with those DEDICATED areas then the data is left in place, otherwise it gets relocated to GENERAL areas (shortly before jumping to the entrypoint). The GENERAL areas are allowed to be:

```
Main  2000000h..2FFC000h (excluding bootstrap at 23FEE00h..23FF000h)
WRAM  3000000h..380F000h (excluding bootstrap at 3FFF600h..3FFF800h)
```

The GENERAL areas may not overlap with another DEDICATED area (eg. ARM7 cannot be relocated to the ARM9+ARM9i+ARM7i areas). Concerning the 16K at 2000000h..2003FFFh: ARM7+ARM9i+ARM7i aren't allow to use that, but ARM9 seems to be relocatable to that address (usually that shouldn't be done as it would destroy some DSi system variables).
ARM9i+ARM7i areas can have "Size" and "ROM offset" set to zero (but still need to have a valid nonzero "RAM Load address").
ARM9/ARM7 entrypoints MUST be within ARM9/ARM7 area respectively (NDS allowed entrypoints being anywhere).

New DSi Header Entries

```
180h 20   Global MBK1..MBK5 Setting, WRAM Slots
194h 12   Local ARM9 MBK6..MBK8 Setting, WRAM Areas
1A0h 12   Local ARM7 MBK6..MBK8 Setting, WRAM Areas
1ACh 3    Global MBK9 Setting, WRAM Slot Write Protect
1AFh 1    Global WRAMCNT Setting (usually 03h) (FCh/00h in SysMenu/Settings)
1B0h 4    Region flags (bit0=JPN, bit1=USA, bit2=EUR, bit3=AUS, bit4=CHN,
            bit5=KOR, bit6-31=Reserved) (FFFFFFFFh=Region Free)
1B4h 4    Access control (AES Key Select)
            bit0 Common Client Key ;want 380F000h=3FFC600h+00h "common key"
            bit1 AES Slot B  ;380F010h=3FFC400h+180h and KEY1=unchanged
            bit2 AES Slot C  ;380F020h=3FFC400h+190h and KEY2.Y=3FFC400h+1A0h
            bit3 SD Card            ;want Device I
            bit4 NAND Access        ;want Device A-H and KEY3=intact
            bit5 Game Card Power On                 ;tested with bit8
            bit6 Shared2 File                       ;used... but WHAT for?
            bit7 Sign JPEG For Launcher (AES Slot B);select 1 of 2 jpeg keys?
            bit8 Game Card NTR Mode                 ;tested with bit5
            bit9 SSL Client Cert (AES Slot A) ;KEY0=3FFC600h+30h (twl-*.der)
            bit10 Sign JPEG For User (AES Slot B) ;\
            bit11 Photo Read Access               ; seems to be unused
            bit12 Photo Write Access              ; (and, usually ZERO,
            bit13 SD Card Read Access             ; even if the stuff is
            bit14 SD Card Write Access            ; accessed)
            bit15 Game Card Save Read Access      ; (bit11 set in flipnote)
            bit16 Game Card Save Write Access     ;/
            bit31 Debugger Common Client Key  ;want 380F000h=3FFC600h+10h
1B8h 4    ARM7 SCFG_EXT7 setting (bit0,1,2,10,18,31)
1BCh 3    Reserved/flags? (zerofilled)
1BFh 1    Flags (usually 01h) (DSiware Browser: 0Bh)
            bit0: TSC Touchscreen/Sound Controller Mode (0=NDS, 1=DSi)
            bit1: Require EULA Agreement
            bit2: Custom Icon  (0=No/Normal, 1=Use banner.sav)
            bit3: Show Nintendo Wi-Fi Connection icon in Launcher
            bit4: Show DS Wireless icon in Launcher
            bit5: NDS cart with icon SHA1  (DSi firmware v1.4 and up)
            bit6: NDS cart with header RSA (DSi firmware v1.0 and up)
            bit7: Developer App
1C0h 4    ARM9i ROM Offset (usually XX03000h, XX=1MB-boundary after NDS area)
1C4h 4    Reserved (zero)
1C8h 4    ARM9i RAM Load address
1CCh 4    ARM9i Size
```

```
1D0h 4    ARM7i ROM Offset
1D4h 4    SD/MMC Device List ARM7 RAM Addr; 400h-byte initialized by firmware
1D8h 4    ARM7i RAM Load address
1DCh 4    ARM7i Size
1E0h 4    Digest NTR region offset (usually same as ARM9 rom offs, 0004000h)
1E4h 4    Digest NTR region length
1E8h 4    Digest TWL region offset (usually same as ARM9i rom offs, XX03000h)
1ECh 4    Digest TWL region length
1F0h 4    Digest Sector Hashtable offset ;\SHA1-HMAC's on all sectors
1F4h 4    Digest Sector Hashtable length ;/in above NTR+TWL regions
1F8h 4    Digest Block Hashtable offset  ;\SHA1-HMAC's on each N entries
1FCh 4    Digest Block Hashtable length  ;/in above Sector Hashtable
200h 4    Digest Sector size       (eg. 400h bytes per sector)
204h 4    Digest Block sectorcount (eg. 20h sectors per block)
208h 4    Icon/Title size (usually 23C0h for DSi) (older 840h-byte works too)
20Ch 1    SD/MMC size of "shared2\0000" file in 32Kbyte units? (dsi sound)
20Dh 1    SD/MMC size of "shared2\0001" file in 32Kbyte units?
              ;or are shared2 sizes rather counted in 16Kbyte cluster units?
20Eh 1    EULA Version (01h)  ?                                    !
20Fh 1    Use Ratings  (00h)  ?                                    !
210h 4    Total Used ROM size, INCLUDING DSi area (optional, can be 0)
214h 1    SD/MMC size of "shared2\0002" file in 32Kbyte units?
215h 1    SD/MMC size of "shared2\0003" file in 32Kbyte units?
216h 1    SD/MMC size of "shared2\0004" file in 32Kbyte units?
217h 1    SD/MMC size of "shared2\0005" file in 32Kbyte units?
218h 4    ARM9i Parameters Table Offset (84 D0 04 00) ???  ;base=[028h]
21Ch 4    ARM7i Parameters Table Offset (2C 05 00 00) ???  ;base=[038h]
220h 4    Modcrypt area 1 offset ;usually same as ARM9i rom offs (XX03000h)
224h 4    Modcrypt area 1 size   ;usually min(4000h,ARM9iSize+Fh AND not Fh)
228h 4    Modcrypt area 2 offset (0=None)
22Ch 4    Modcrypt area 2 size   (0=None)
230h 4    Title ID, Emagcode (aka Gamecode spelled backwards)
234h 1    Title ID, Filetype (00h=Cartridge, 04h=DSiware, 05h=System Fun
            Tools, [0Fh=Non-executable datafile without cart header],
            15h=System Base Tools, 17h=System Menu)
235h 1    Title ID, Zero     (00h=Normal)
236h 1    Title ID, Three    (03h=DSi) (as opposed to Wii or 3DS)
237h 1    Title ID, Zero     (00h=Normal)
238h 4    SD/MMC (DSiware) "public.sav" filesize in bytes  (0=none)
23Ch 4    SD/MMC (DSiware) "private.sav" filesize in bytes (0=none)
240h 176  Reserved (zero-filled)
```

Parental Control Age Ratings (set all entries to 80h to allow any age)

```
2F0h 10h  Parental Control Age Ratings (for different countries/areas)
           Bit7: Rating exists for local country/area
           Bit6: Game is prohibited in local country/area?
           Bit5: Unused
           Bit4-0: Age rating for local country/area (years)
2F0h 1    CERO (Japan)       (0=None/A, 12=B, 15=C, 17=D, 18=Z)
2F1h 1    ESRB (US/Canada)   (0=None, 3=EC, 6=E, 10=E10+, 13=T, 17=M)
2F2h 1    Reserved           (0=None)
2F3h 1    USK (Germany)      (0=None, 6=6+, 12=12+, 16=16+, 18=18+)
2F4h 1    PEGI (Pan-Europe)  (0=None, 3=3+, 7=7+, 12=12+, 16=16+, 18=18+)
2F5h 1    Reserved           (0=None)
2F6h 1    PEGI (Portugal)    (0=None, 4=4+, 6=6+, 12=12+, 16=16+, 18=18+)
2F7h 1    PEGI and BBFC (UK) (0=None, 3, 4=4+/U, 7, 8=8+/PG, 12, 15, 16, 18)
2F8h 1    AGCB (Australia)   (0=None/G, 7=PG, 14=M, 15=MA15+, plus 18=R18+?)
2F9h 1    GRB (South Korea)  (0=None, 12=12+, 15=15+, 18=18+)
2FAh 6    Reserved (6x)      (0=None)
N/A? -    DEJUS (Brazil) (L, 10, 12, 14, 16, 18)
N/A? -    GSRMR (Taiwan) (formerly CSRR) (0,6,12,18) (and GSRMR: 15)
N/A? -    PEGI (Finland) (discontinued 2007, shortly before DSi launch)
           bit0-4 Rating (0..18)
           bit6   Pending
           bit7   Enabled
```

SHA1-HMAC's and RSA-SHA1

```
  300h 20   SHA1-HMAC hash ARM9 (with encrypted secure area)     ;[020h,02Ch]
  314h 20   SHA1-HMAC hash ARM7                                  ;[030h,03Ch]
  328h 20   SHA1-HMAC hash Digest master                         ;[1F8h,1FCh]
  33Ch 20   SHA1-HMAC hash Icon/Title (also in newer NDS titles) ;[068h,208h]
  350h 20   SHA1-HMAC hash ARM9i (decrypted)                     ;[1C0h,1CCh]
  364h 20   SHA1-HMAC hash ARM7i (decrypted)                     ;[1D0h,1DCh]
  378h 20   Reserved (zero-filled) (but used for non-whitelisted NDS titles)
  38Ch 20   Reserved (zero-filled) (but used for non-whitelisted NDS titles)
  3A0h 20   SHA1-HMAC hash ARM9 (without 16Kbyte secure area)    ;[020h,02Ch]
  3B4h 2636 Reserved (zero-filled)
  E00h 180h Reserved and unchecked region, always zero. Used for passing
              arguments in debug environment.
  F80h 80h  RSA-SHA1 signature across header entries [000h..DFFh]
```

Reserved Area

```
  1000h..3FFFh  Non-Load area in ROMs... but contains sth in DSiWare files!?!
```

DSiware/System Utilities

Files saved on SD card or internal eMMC memory are having the same header as ROM carts, with some differences:

```
  No need for NDS backwards compatibility (since DSiware is DSi only)
  Entry 3A0h can be zero-filled (in LAUNCHER)
```

DSiware files are usually marked as [012h]=03h=DSi (exceptions are the DS Download Play and PictoChat utilities, which are marked [012h]=00h=NDS, since they are actually running in NDS mode (yet having at least two DSi-specific features: 1st is allowing to reboot into DSi menu via Powerman SPI, 2nd is requiring enabled wifi channels in 2FFFCFAh)).

SHA1-HMAC

The SHA1-HMAC's in cart header and Digest tables are SHA1 checksums with a 40h-byte HMAC key (values 21h, 06h, C0h, DEh, BAh, ..., 24h), the key is contained in the Launcher, and it's also stored in most DSi cartridges (probably used for verifying Digest values when loading additional data after booting). The key can be used for verifying checksums, but (due to the RSA signature) not for changing them. See BIOS chapter for SHA1/HMAC pseudo code.

RSA-SHA1

The RSA-SHA1 value is a normal SHA1 (not SHA1-HMAC) across header entries [000h..DFFh], the 20-byte value is padded to 127-byte size (01h, 105xFFh, 00h, followed by the 20 SHA1 bytes). It can be decrypted (via SWI 22h) using the 80h-byte RSA public keys located in ARM9BIOS (note that there are at least four different RSA keys, one is used for games, and others for system files), and can be then verfied against the SHA1 checksum (computed via SWI 27h).

BIOS RSA Functions (DSi only)

The private key needed for encryption is unknown, which is unfortunately preventing to boot unlicensed (homebrew) software.

Modcrypt (AES-CTR) (optional, carthdr[220h..22Fh] can be all zero)

Modcrypt is a new additional way of encrypting parts of the NDS ROM executable binary modules using AES CTR. It is mostly being used to encrypt the ARM9i and ARM7i binaries. DSi cartridges are usually having only the ARM9i binary encrypted (as area 1), while NAND based applications have both the ARM9i and ARM7i binaries encrypted (as area 1 and 2).

The initial AES Counter value (IV) is:

```
  Modcrypt Area 1 IV[0..F]: First 16 bytes of the ARM9 SHA1-HMAC   [300h..30Fh]
  Modcrypt Area 2 IV[0..F]: First 16 bytes of the ARM7 SHA1-HMAC   [314h..323h]
```

The AES key depends of flags in the cartridge header:

```
 IF header[01Ch].Bit1=0
  None (modcrypt disabled)
 ELSEIF header[01Ch].Bit2 OR header[1BFh].Bit7 THEN (probably for prototypes)
  Debug KEY[0..F]: First 16 bytes of the header                   [000h..00Fh]
 ELSE (commonly used for retail software)
```

```
  Retail KEY_X[0..7]: Fixed 8-byte ASCII string                   ("Nintendo")
  Retail KEY_X[8..B]: The 4-byte gamecode, forwards               [00Ch..00Fh]
  Retail KEY_X[C..F]: The 4-byte gamecode, backwards              [00Fh..00Ch]
  Retail KEY_Y[0..F]: First 16 bytes of the ARM9i SHA1-HMAC       [350h..35Fh]
```

Above does describe how modcrypted areas should look like. However, there are several circumstances where the firmware can't actually decrypt that areas...
Theoretically, the modcrypt areas can span over any of the ARM9i/ARM7i and ARM9/ARM7 areas (in practice, cartridges should never use modcrypt for the ARM9/ARM7 areas because NDS consoles would leave them undecrypted; that restriction doesn't apply to DSiware though).
Theoretically, a large modcrypt area could contain several data areas, but the launcher does decrypt only the first matching data area (areas are processed in order ARM9, ARM7, ARM9i, ARM7i). Moreover, matching data areas must be INSIDE of the modcrypt area (ie. the data area must be same size, or smaller than the modrypt area) (whereas, the launcher is weirdly rounding-up the data size to a multiple of 20h-bytes when checking that matches) (as a side-effect, each data area can be decrypted starting with "IV+0", rather than needing "IV+(offset_within_modcrypt_area/10h").
Theoretically, AES decryption could be done byte-wise, however, the AES hardware is doing it in 10h-byte chunks, unknown if the launcher does require 10h-byte alignments; however, in fact, the launcher seems to be rounding the modcrypt END address to 20h-byte boundary - so it's safest to stick with 20h-byte aligned start+size values for modcrypt areas, as well as for corresponding data areas.
Note: The size of the modcrypt areas can exceed the AES hardware's size limit of max FFFF0h bytes (eg. in DSi Sound utility), in such cases decryption must be split to smaller AES chunks; with manually increased IV.
Modcrypt area 1 and 2 can overlap each other (whereas, it doesn't matter which of them is processed first, since the encryption/decryption is done by XORing).
Note: The ARM9 code for modcrypt/data areas is at 26A6BB8h in Launcher v1.4E.

Digests (optional, carthdr[1E0h..207h] can be all zero)
The NDS format has been extended with a hash tree to verify the entire contents of an NDS ROM. The NDS ROM is divided into sectors, and each sector will be hashed and have its hash stored in the digest sector hashtable. The size of a sector is defined in the header aswell. Furthermore, the sector hashtable is partitioned and hashed again to form block hashes. This block hashtable is hashed again into a single hash called the digest master hash. These hashtables can be used to verify that the sectors of a NDS ROM have not been tampered with, since the integrity of a sector hash can be verified by a block hash, which in turn can be verified by the master hash. And this hash is part of the header, which is signed with RSA.
The sector hashtable reaches over the NTR and TWL regions, respectively.

Cartridge Protocol
The DSi cartridge protocol is same as on NDS; with one new command (3Dh) for unlocking DSi specific memory regions. For details,
DS Cartridge Protocol

DSi Touchscreen/Sound Controller

DSi Touchscreen Access
DSi Touchscreen/Sound Init Flowcharts

AIC3000D Registers
DSi TSC, Register Summary
DSi TSC[0:00h..1Ah], Basic PLL and Timing Control
DSi TSC[0:1Bh..23h], Codec Control
DSi TSC[0:24h..32h], Status and Interrupt Flags
DSi TSC[0:33h..3Bh], Pin Control
DSi TSC[0:3Ch..55h], DAC/ADC and Beep
DSi TSC[0:56h..7Fh], AGC and ADC

DSi TSC[1:xxh], DAC and ADC Routing, PGA, Power-Controls and MISC Logic
DSi TSC[3:xxh], Touchscreen/SAR Control and TSC[FCh:xxh], Buffer
DSi TSC[04h..05h:xxh], ADC Digital Filter Coefficient RAM
DSi TSC[08h..0Fh:xxh], DAC Digital Filter Coefficient RAM
DSi TSC[20h..2Bh:xxh], TSC[40h..5Fh:xxh] ADC/DAC Instruction RAM

DSi Touchscreen Access

The Touch Screen Controller (for lower LCD screen) is accessed via SPI bus,
DS Serial Peripheral Interface Bus (SPI)
so far, it's same as on NDS, but the SPI touchscreen commands are having an entirely different format in DSi mode:
The DSi touchscreen registers are selected via a combination of a PAGE byte and an INDEX byte. The PAGE byte is located at INDEX=00h, and it does somewhat 'bankswitch' the contents of INDEX=01h..7Fh. And INDEX can be incremented manually, or automatically (but, confusingly, the manual increment doesn't work for reading Y coordinates).
SPI clock should be set to 4MHz for DSi Mode touchscreen access (unlike NDS, which used 2MHz). The PENIRQ bit in port 4000136h is always zero in DSi mode.
When reading data: Write dummy 00h-bytes in output direction.

AIC3000D

DSi Touchscreen PAGE:INDEX values
The INDEX/Direction byte is written as first byte after SPI chip select:
```
0     Direction for following data bytes (0=Write, 1=Read)
1-7   INDEX (00h..7Fh) for following data bytes (auto-increasing)
```
However, the chip has more than 128 registers, stored in multiple pages.
```
TSC[00h]=PAGE          ;<-- change page (at INDEX=0)
TSC[PAGE:INDEX]        ;<-- access registers in select page
```

Pen Down Testing
```
if (TSC[3:09h] AND 40h)<>0 then return(not_pressed)   ;ADC Ready Flag
if (TSC[3:0Eh] AND 03h)<>0 then return(not_pressed)   ;Undocumented Flags?
return(pressed)
```
Note: On NDS, this would be done by reading port 4000136h.bit6, which isn't supported in DSi mode.

X/Y Coordinate Reading
```
touchdata[0..19] = TSC[FCh:01h..14h]     ;read page FCh, index(1..20)
rawx=0, rawy=0
for i=0 to 8 step 2
  x = touchdata[i+0]*100h+touchdata[i+1]
  y = touchdata[i+10]*100h+touchdata[i+11]
  if (x or y) and F000h then return(not_pressed)
  rawx=rawx+x, rawy=rawy+y
return(rawx/5, rawy/5)
```
The resulting 12bit coordinates are same as on NDS (ie. they need to be further processed using the Calibration Points from User Settings).

Touchscreen X/Y Coordinates
```
0-11   Coordinate (0..FFFh) (usually 000h when not pressed)
12-14  State (0=Pressed, 7=Released) (or sometimes also 1 or 3=Released)
15     State Changed (0=No, 1=Newly pressed/released; cleared after read)
```
Bit12-14 are usually set to 7 when releasing the screen (though sometimes they become 1 or 3 when releasing the screen, and do stay so until newly pressing it).
Bit15 is cleared after reading (so it will be usually seen only in the first MSB, ie. at INDEX=01h) (though

maybe it can also occur elsewhere if it becomes newly set during the SPI transfer).

Microphone
The microphone input was part of the TSC on NDS. In DSi mode it is reportedly somehow changed, using a new "CODEC" (whatever that means). Maybe it's accessed directly via an ARM7 port (and/or TEAK port?), instead of via SPI bus?

NDS Backwards Compatibility Mode
The DSi hardware can emulate the NDS-style touchscreen protocol (with X/Y/MIC channels and with additional PENIRQ flag; but without Pressure or Temperature channels).
DS Touch Screen Controller (TSC)
DSi enhanced games should implement both modes, and use the new mode on DSi consoles, and the old mode on NDS consoles (alternately, CartHdr[1BFh].bit0=0 will activate the old mode for both NDS+DSi).
Unknown how to activate that backwards compatibility mode (might be done via some Touchscreen SPI register, or maybe some Powerman SPI register).
If the backwards compatibility mode isn't enabled, then trying to read the touchscreen in NDS fashion will return nothing but zeroes for all TSC channels (and also zero for the PENIRQ bit).

DSi Touchscreen/Sound Init Flowcharts

touch_enable_tsc_init_list:
```
TSC[3:0Eh]=00h  ;Undoc (RMW: bit7=0)
TSC[3:02h]=18h  ;SAR ADC clk divider, Div8 (12bit mode) (RMW bit34=3, bit7=?)
TSC[3:0Fh]=A0h  ;Scan Mode Timer
TSC[3:0Eh].28h  ;Undoc (RNW: bit345=5)
TSC[3:0Eh].28h  ;Undoc (RMW: bit6=0)
TSC[3:03h]=87h  ;SAR ADC Control 2 (SelfByPenDown, ScanXY, /PENIRQ)
TSC[3:05h].04h  ;Stabilization time = 30us (RMW bit012=4)
TSC[3:04h].02h  ;Sense time = 3us          (RMW bit012=2)
TSC[3:04h].22h  ;Precharge time = 3us      (RMW bit456=2)
TSC[3:12h].00h  ;Debounce Time = 0us       (RMW bit012=0)
TSC[3:0Eh].A8h  ;Undoc                     (RMW bit7=1)
```

mic_enable_tsc_init_list:
This is needed to unmute the microphone (otherwise microphone will return only 0000h or FFFFh.
```
TSC[1:2Eh]=03h  ;MICBIAS=AVDD
TSC[0:51h]=80h  ;ADC Digital Mic, on
TSC[0:52h]=00h  ;ADC Digital Volume Control Fine Adjust, unmute
TSC[1:2Fh]=37h  ;MIC PGA=27.5dB (or use other value, if desired)
```

mic_disable_tsc_init_list:
```
TSC[0:52h]=80h  ;ADC Digital Volume Control Fine Adjust, mute
TSC[0:51h]=00h  ;ADC Digital Mic, off
TSC[1:2Eh]=00h  ;MICBIAS=Off
```

basic_tsc_init_list:
This is done by DSi Launcher/Unlaunch (and other titles usually won't need to do this.
```
TSC[0:01h]=01h  ;Software Reset
TSC[0:39h]=66h  ;ADC DC Measurement 1
TSC[1:20h]=16h  ;Class-D Speaker Amplifier (RMW:bit4=1)
TSC[0:04h]=00h  ;Clock-Gen Muxing
TSC[0:12h]=81h  ;ADC NADC Value
TSC[0:13h]=82h  ;ADC MADC Value
TSC[0:51h]=82h  ;ADC Digital Mic
TSC[0:51h]=00h  ;ADC Digital Mic again
TSC[0:04h]=03h  ;Clock-Gen Muxing
TSC[0:05h]=A1h  ;PLL P and R-Values
```

```
TSC[0:06h]=15h  ;PLL J-Value
TSC[0:0Bh]=87h  ;DAC NDAC Value
TSC[0:0Ch]=83h  ;DAC MDAC Value
TSC[0:12h]=87h  ;ADC NADC Value
TSC[0:13h]=83h  ;ADC MADC Value
TSC[3:10h]=88h  ;Scan Mode Timer Clock (RMW:bit0-6)
TSC[4:08h..0Dh]=7Fh,E1h,80h,1Fh,7Fh,C1h  ;some coeff's
TSC[0:41h]=08h  ;DAC Left Volume Control
TSC[0:42h]=08h  ;DAC Right Volume Control
TSC[0:3Ah]=00h  ;GPI3 Pin Control
TSC[4:08h..0Dh]=7Fh,E1h,80h,1Fh,7Fh,C1h  ;some coeff's ;again?
TSC[1:2Fh]=2Bh  ;MIC PGA
TSC[1:30h]=40h  ;P-Terminal Delta-Sigma Mono ADC Channel Fine-Gain Input
TSC[1:31h]=40h  ;M-Terminal ADC Input Selection
TSC[1:32h]=60h  ;Input CM Settings
TSC[0:74h]=82h  ;VOL/MICDET-Pin SAR ADC - Volume Control (RMW)
TSC[0:74h]=92h  ;VOL/MICDET-Pin SAR ADC - Volume Control (RMW)
TSC[0:74h]=D2h  ;VOL/MICDET-Pin SAR ADC - Volume Control (RMW)
TSC[1:21h]=20h  ;HP Output Drivers POP Removal Settings
TSC[1:22h]=F0h  ;Output Driver PGA Ramp-Down Period Control
TSC[0:3Fh]=D4h  ;DAC Data-Path Setup (RMW)
TSC[1:23h]=44h  ;DAC_L and DAC_R Output Mixer Routing
TSC[1:1Fh]=D4h  ;Headphone Drivers
TSC[1:28h]=4Eh  ;HPL Driver (Left Headphone)
TSC[1:29h]=4Eh  ;HPR Driver (Right Headphone)
TSC[1:24h]=9Eh  ;Analog Volume to HPL (Left Headphone)
TSC[1:25h]=9Eh  ;Analog Volume to HPR (Right Headphone)
TSC[1:20h]=D4h  ;Class-D Speaker Amplifier
TSC[1:2Ah]=14h  ;SPL Driver (Left Speaker)
TSC[1:2Bh]=14h  ;SPR Driver (Right Speaker)
TSC[1:26h]=A7h  ;Analog Volume to SPL (Left Speaker)
TSC[1:27h]=A7h  ;Analog Volume to SPR (Right Speaker)
TSC[0:40h]=00h  ;DAC Volume Control
(should set DSi.GPIO.data.bit7 here, but can be also done elsewhere)
TSC[0:3Ah]=60h  ;GPI3 Pin Control
```

nds_mode_tsc_init_list:
Switches to NDS Touchscreen/sound mode. Usually done when starting NDS titles (or DSi titles that request NDS-TSC mode in carthdr[1BFh]).

```
TSC[1:26h]=ACh  ;\special setting (when found special gamecode)
TSC[1:27h]=ACh  ;/
TSC[1:26h]=A7h  ;\normal setting (for any other gamecodes)
TSC[1:27h]=A7h  ;/
TSC[1:2Eh]=03h  ;MICBIAS=AVDD
TSC[3:03h]=00h  ;SAR ADC Control 2
TSC[1:21h]=20h  ;HP Output Drivers POP Removal Settings
TSC[1:22h]=F0h  ;Output Driver PGA Ramp-Down Period Control (70h OR 80h)
TSC[1:22h]=70h  ;Output Driver PGA Ramp-Down Period Control (bit7=0)
TSC[0:52h]=80h  ;ADC Digital Volume Control Fine Adjust
TSC[0:51h]=00h  ;ADC Digital Mic
READ[3:02h] (returns 00h)
TSC[3:02h].Bit7=1 ;SAR ADC Control 1 (set to 80h) (or 98h?) (RMW)
TSC[FFh:05h]=00h ;TSC final enter NDS mode
```

And, set powerman[0].bit0=1 (bit0=sound amplifier on) (this is actually part of TSC chip, accessed via "POWERMAN" SPI chipselect signal).

DSi TSC, Register Summary

The DSi's Touchscreen/Sound controller (AIC3000D) is essentially a Texas Instruments TSC2117 chip (possibly with some customizations for NDS backwards compatibility mode).

TSC[page:index] registers are accessed via SPI bus with 15bit address space:

```
  7bit index: selected via the first SPI byte, with direction flag in bit0
  8bit page:  selected by writing to index 00h, ie. to TSC[xxh:00h]
```

TSC page select (for "TSC[page:index]" addressing)

```
  TSC[xxh:00h] - Page Select Register (00h)
```

TSC Basic PLL and Timing Control

```
  TSC[0:01h] - Software Reset (00h)
  TSC[0:02h] - Reserved (xxh) (R)
  TSC[0:03h] - Overtemperature OT Flag (02h..FFh) (R)
  TSC[0:04h] - Clock-Gen Muxing (00h)
  TSC[0:05h] - PLL P and R-Values (11h)
  TSC[0:06h] - PLL J-Value (04h)
  TSC[0:07h,08h] - PLL D-Value MSB,LSB (0000h)
  TSC[0:09h,0Ah] - Reserved (xxh)
  TSC[0:0Bh] - DAC NDAC Value (01h)
  TSC[0:0Ch] - DAC MDAC Value (01h)
  TSC[0:0Dh,0Eh] - DAC DOSR Value MSB,LSB (0080h)
  TSC[0:0Fh] - DAC IDAC Value (80h)
  TSC[0:10h] - DAC miniDSP Engine Interpolation (08h)
  TSC[0:11h] - Reserved (xxh)
  TSC[0:12h] - ADC NADC Value (01h)
  TSC[0:13h] - ADC MADC Value (01h)
  TSC[0:14h] - ADC AOSR Value (80h)
  TSC[0:15h] - ADC IADC Value (80h)
  TSC[0:16h] - ADC miniDSP Engine Decimation (04h)
  TSC[0:17h,18h] - Reserved (xxh)
  TSC[0:19h] - CLKOUT MUX (00h)
  TSC[0:1Ah] - CLKOUT Divider M Value (01h)
```

TSC Codec Control

```
  TSC[0:1Bh] - Codec Interface Control 1 (00h) (R/W)
  TSC[0:1Ch] - Data-Slot Offset Programmability (00h)
  TSC[0:1Dh] - Codec Interface Control 2 (00h)
  TSC[0:1Eh] - BCLK Divider N Value (01h)
  TSC[0:1Fh] - Codec Secondary Interface Control 1 (00h)
  TSC[0:20h] - Codec Secondary Interface Control 2 (00h)
  TSC[0:21h] - Codec Secondary Interface Control 3 (00h)
  TSC[0:22h] - I2C Bus Condition (00h)
  TSC[0:23h] - Reserved (xxh)
```

TSC Status and Interrupt Flags

```
  TSC[0:24h] - ADC Flag Register (0xh) (R)
  TSC[0:25h] - DAC Flag Register (00h) (R)
  TSC[0:26h] - DAC Flag Register (00h) (R)
  TSC[0:27h] - Overflow Flags (00h) (R)
  TSC[0:28h..2Bh] - Reserved (xxh)
  TSC[0:2Ch] - Interrupt Flags DAC, sticky (00h..30h) (R)
  TSC[0:2Dh] - Interrupt Flags ADC, sticky (00h..18h) (R)
  TSC[0:2Eh] - Interrupt Flags DAC, non-sticky? (00h..30h) (R)
  TSC[0:2Fh] - Interrupt Flags ADC, non-sticky? (00h..18h) (R)
  TSC[0:30h] - INT1 Control Register (Select INT1 Sources) (00h)
  TSC[0:31h] - INT2 Control Register (Select INT2 Sources) (00h)
  TSC[0:32h] - INT1 and INT2 Control Register (00h)
```

TSC Pin Control

```
  TSC[0:33h] - GPIO1 In/Out Pin Control (00h..C2h)
  TSC[0:34h] - GPIO2 In/Out Pin Control (00h..C2h)
  TSC[0:35h] - SDOUT (OUT Pin) Control (12h)
  TSC[0:36h] - SDIN (IN Pin) Control (02h or 03h)
  TSC[0:37h] - MISO (OUT Pin) Control (02h)
  TSC[0:38h] - SCLK (IN Pin) Control (02h..03h)
  TSC[0:39h] - GPI1 and GPI2 Pin Control (00h..11h)
  TSC[0:3Ah] - GPI3 Pin Control (00h..10h)
  TSC[0:3Bh] - Reserved (xxh)
```

TSC DAC/ADC and Beep

TSC[0:3Ch] - DAC Instruction Set (01h)
TSC[0:3Dh] - ADC Instruction Set (04h)
TSC[0:3Eh] - Programmable Instruction Mode-Control Bits (00h)
TSC[0:3Fh] - DAC Data-Path Setup (14h)
TSC[0:40h] - DAC Volume Control (0Ch)
TSC[0:41h] - DAC Left Volume Control (00h)
TSC[0:42h] - DAC Right Volume Control (00h)
TSC[0:43h] - Headset Detection (00h..60h)
TSC[0:44h] - DRC Control 1 (0Fh)
TSC[0:45h] - DRC Control 2 (38h)
TSC[0:46h] - DRC Control 3 (00h)
TSC[0:47h] - Beep Generator and Left Beep Volume (00h)
TSC[0:48h] - Beep Generator and Right Beep Volume (00h)
TSC[0:49h,4Ah,4Bh] - Beep Length MSB,MID,LSB (0000EEh)
TSC[0:4Ch,4Dh] - Beep Frequency Sin(x) MSB,LSB (10D8h)
TSC[0:4Eh,4Fh] - Beep Frequency Cos(x) MSB,LSB (7EE3h)
TSC[0:50h] - Reserved (xxh)
TSC[0:51h] - ADC Digital Mic (00h)
TSC[0:52h] - ADC Digital Volume Control Fine Adjust (80h)
TSC[0:53h] - ADC Digital Volume Control Coarse Adjust (00h)
TSC[0:54h,55h] - Reserved (xxh)

TSC AGC and ADC (Auto-Gain-Control & Analog-to-Digital Converter)

TSC[0:56h] - AGC Control 1 (00h)
TSC[0:57h] - AGC Control 2 (00h)
TSC[0:58h] - AGC Maximum Gain (7Fh, uh that's 7Fh=Reserved?)
TSC[0:59h] - AGC Attack Time (00h)
TSC[0:5Ah] - AGC Decay Time (00h)
TSC[0:5Bh] - AGC Noise Debounce (00h)
TSC[0:5Ch] - AGC Signal Debounce (00h)
TSC[0:5Dh] - AGC Gain-Applied Reading (xxh) (R)
TSC[0:5Eh...65h] - Reserved (xxh)
TSC[0:66h] - ADC DC Measurement 1 (00h)
TSC[0:67h] - ADC DC Measurement 2 (00h)
TSC[0:68h,69h,6Ah] - ADC DC Measurement Output MSB,MID,LSB (R) (000000h)
TSC[0:6Bh...73h] - Reserved (xxh)
TSC[0:74h] - VOL/MICDET-Pin SAR ADC - Volume Control (00h)
TSC[0:75h] - VOL/MICDET-Pin Gain (xxh) (R)
TSC[0:76h...7Fh] - Reserved (xxh)

TSC TSC, DAC and ADC Routing, PGA, Power-Controls and MISC Logic

TSC[1:01h..1Dh] - Reserved (xxh)
TSC[1:1Eh] - Headphone and Speaker Amplifier Error Control (00h)
TSC[1:1Fh] - Headphone Drivers (04h)
TSC[1:20h] - Class-D Speaker Amplifier (06h)
TSC[1:21h] - HP Output Drivers POP Removal Settings (3Eh)
TSC[1:22h] - Output Driver PGA Ramp-Down Period Control (00h)
TSC[1:23h] - DAC_L and DAC_R Output Mixer Routing (00h)
TSC[1:24h] - Analog Volume to HPL (Left Headphone) (7Fh)
TSC[1:25h] - Analog Volume to HPR (Right Headphone) (7Fh)
TSC[1:26h] - Analog Volume to SPL (Left Speaker) (7Fh)
TSC[1:27h] - Analog Volume to SPR (Right Speaker) (7Fh)
TSC[1:28h] - HPL Driver (Left Headphone) (02h)
TSC[1:29h] - HPR Driver (Right Headphone) (02h)
TSC[1:2Ah] - SPL Driver (Left Speaker) (00h)
TSC[1:2Bh] - SPR Driver (Right Speaker) (00h)
TSC[1:2Ch] - HP Driver Control (00h)
TSC[1:2Dh] - Reserved (xxh)
TSC[1:2Eh] - MICBIAS (00h)
TSC[1:2Fh] - MIC PGA (80h)
TSC[1:30h] - P-Terminal Delta-Sigma Mono ADC Channel Fine-Gain Input (00h)
TSC[1:31h] - M-Terminal ADC Input Selection (00h)
TSC[1:32h] - Input CM Settings (00h)
TSC[1:33h..FFh] - Reserved (xxh)

Reserved Page

TSC[2:01h..FFh] - Reserved (00h)

TSC Touchscreen/SAR Control

TSC[3:01h] - Reserved (xxh)
TSC[3:02h] - SAR ADC Control 1 (00h)
TSC[3:03h] - SAR ADC Control 2 (00h)
TSC[3:04h] - Precharge and Sense (00h)
TSC[3:05h] - Panel Voltage Stabilization (00h)
TSC[3:06h] - Voltage Reference (20h)
TSC[3:07h,08h] - Reserved (xxh)
TSC[3:09h] - Status Bits 1 (40h) (R)
TSC[3:0Ah] - Status Bits 2 (00h) (R)
TSC[3:0Bh,0Ch] - Reserved (xxh)
TSC[3:0Dh] - Buffer Mode (03h) ;DSi: Unused,seems to use TSC[3:0Eh] instead
TSC[3:0Eh] - Reserved / Undocumented (read by DSi for Pen Down Test) (0Fh)
TSC[3:0Fh] - Scan Mode Timer (40h)
TSC[3:10h] - Scan Mode Timer Clock (81h)
TSC[3:11h] - SAR ADC Clock (81h)
TSC[3:12h] - Debounce Time for Pen-Up Detection (00h)
TSC[3:13h] - Auto AUX Measurement Selection (00h)
TSC[3:14h] - Touch-Screen Pen Down (00h)
TSC[3:15h] - Threshold Check Flags Register (00h) (R)
TSC[3:16h,17h] - AUX1 Maximum Value Check MSB,LSB (0000h)
TSC[3:18h,19h] - AUX1 Minimum Value Check MSB,LSB (0000h)
TSC[3:1Ah,1Bh] - AUX2 Maximum Value Check MSB,LSB (0000h)
TSC[3:1Ch,1Dh] - AUX2 Minimum Value Check MSB,LSB (0000h)
TSC[3:1Eh,1Fh] - Temperature(TEMP1/TEMP2) Maximum Value Check MSB,LSB (0000h)
TSC[3:20h,21h] - Temperature(TEMP1/TEMP2) Minimum Value Check MSB,LSB (0000h)
TSC[3:22h...29h] - Reserved (xxh)
TSC[3:2Ah,2Bh] - Touchscreen X-Coordinate Data MSB,LSB (0000h) (R)
TSC[3:2Ch,2Dh] - Touchscreen Y-Coordinate Data MSB,LSB (0000h) (R)
TSC[3:2Eh,2Fh] - Touchscreen Z1-Pressure Register MSB,LSB (0000h) (R)
TSC[3:30h,31h] - Touchscreen Z2-Pressure Register MSB,LSB (0000h) (R)
TSC[3:32h...35h] - Reserved (xxh)
TSC[3:36h,37h] - AUX1 Data MSB,LSB (0000h) (R)
TSC[3:38h,39h] - AUX2 Data MSB,LSB (0000h) (R)
TSC[3:3Ah,3Bh] - VBAT Data MSB,LSB (0000h) (R)
TSC[3:3Ch...41h] - Reserved (xxh)
TSC[3:42h,43h] - Temperature TEMP1 Data Register MSB,LSB (0000h) (R)
TSC[3:44h,45h] - Temperature TEMP2 Data Register MSB,LSB (0000h) (R)
TSC[3:46h...7Fh] - Reserved (xxh)

TSC Coefficient RAM and Instruction RAM for ADC/DAC

TSC[04h..05h:xxh] - ADC Coefficient RAM (126 x 16bit)
TSC[06h..07h:xxh] - Reserved (00h)
TSC[08h:01h]   - DAC Coefficient RAM Control (00h)
TSC[08h..0Bh:xxh] - DAC Coefficient RAM, DAC Buffer A (252 x 16bit)
TSC[0Ch..0Fh:xxh] - DAC Coefficient RAM, DAC Buffer B (252 x 16bit)
TSC[10h..1Fh:xxh] - Reserved (00h)
TSC[20h..2Bh:xxh] - ADC DSP Engine Instruction RAM (384 x 24bit)
TSC[2Ch..3Fh:xxh] - Reserved (00h)
TSC[40h..5Fh:xxh] - DAC DSP Engine Instruction RAM (1024 x 24bit)
TSC[60h..FBh:xxh] - Reserved (00h)

TSC Additional 3DS Registers (3DS onlly, and only on the 2nd SPI bus)

TSC[64h:01h..xxh] - 3DS Config Registers for Sound (and Microphone?)
TSC[65h:01h..xxh] - 3DS Config Registers for Sound (and Microphone?)
TSC[67h:01h..xxh] - 3DS Config Registers for Touchscreen and Circle Pad
TSC[FBh:01h..xxh] - 3DS Buffer Mode Data for Touchscreen and Circle Pad

TSC Touchscreen/SAR Buffer

TSC[FCh:01h..xxh] - Buffer Mode Data MSB,LSB (xxxxh) (R)
TSC[FCh:xxh..7Fh] - Reserved (xxh)

TSC Undocumented Registers

TSC[FDh:xxh] - Contains some undocumented non-zero values (DSi specific?)
TSC[FEh:xxh] - Reserved (00h)
TSC[FFh:xxh] - Accessing this page changes operation (DSi specific?)

DSi TSC[0:00h..1Ah], Basic PLL and Timing Control

TSC[xxh:00h] - Page Select Register (00h)

```
7-0   Page Select (00h..FEh) (FFh=Undocumented, enter special mode?)
```

Selects the "page" for the TSC[page:index] addresses.

TSC[0:01h] - Software Reset (00h)

```
7-1   Reserved. Write only zeros to these bits.
0     Software Reset (0=No change, 1=Reset)
```

TSC[0:02h] - Reserved (xxh) (R)

```
7-0   Reserved. Do not write to this register.
```

TSC[0:03h] - Overtemperature OT Flag (02h..FFh) (R)

```
7-2   Reserved. Do not write to these bits. (R)
1     Overtemperature protection flag (0=Alert, 1=Normal) (R)
0     Reserved. Do not write to these bits. (R/W?)
```

Bit1 is valid only if speaker amplifier is powered up.

TSC[0:04h] - Clock-Gen Muxing (00h)

```
7-4   Reserved. Write only zeros to these bits.
3-2   Select PLL_CLKIN   (0=MCLK, 1=BCLK, 2=GPIO1, 3=SDIN)
1-0   Select CODEC_CLKIN (0=MCLK, 1=BCLK, 2=GPIO1, 3=PLL_CLK)
```

See Section 5.8 for more details on clock generation mutiplexing and dividers.

TSC[0:05h] - PLL P and R-Values (11h)

```
7     PLL Enable        (0=Power down, 1=Power up)
6-4   PLL Divider P     (1..7=Div1..7, or 0=Div8)
3-0   PLL Multiplier R  (1..15=Mul1..15, or 0=Mul16)
```

TSC[0:06h] - PLL J-Value (04h)

```
7-6   Reserved. Write only zeros to these bits.
5-0   PLL Multiplier J  (1..63=Mul1..63, or 0=Reserved)
```

TSC[0:07h,08h] - PLL D-Value MSB,LSB (0000h)

```
15-14 Reserved. Write only zeros to these bits.
13-0  PLL fractional multiplier D-Val (14bit)
```

Note that LSB register must be written to immediately after writing to MSB.

TSC[0:09h,0Ah] - Reserved (xxh)

```
7-0   Reserved. Write only zeros to these bits.
```

TSC[0:0Bh] - DAC NDAC Value (01h)

```
7     DAC NDAC Divider Enable (0=Power down, 1=Power up)
6-0   DAC NDAC Divider (1..127=Div1..127, or 0=Div128)
```

TSC[0:0Ch] - DAC MDAC Value (01h)

```
7     DAC MDAC Divider Enable (0=Power down, 1=Power up)
6-0   DAC MDAC Divider (1..127=Div1..127, or 0=Div128)
```

TSC[0:0Dh,0Eh] - DAC DOSR Value MSB,LSB (0080h)

```
15-10 Reserved
9-0   DAC OSR value "DOSR" (1..1023, or 0=1024)
```

DOSR should be an integral multiple of the interpolation ratio in TSC[0:10h].
Note that LSB register must be written to immediately after writing to MSB.

TSC[0:0Fh] - DAC IDAC Value (80h)

```
7-0   Number of instructions for DAC miniDSP engine (IDAC=N*4)
      (1..255 = 4..1020 (N*4), or 0=1024)
```

IDAC should be an integral multiple of the interpolation ratio in TSC[0:10h].

TSC[0:10h] - DAC miniDSP Engine Interpolation (08h)

```
7-4   Reserved. Do not write to these registers.
3-0   Interpolation ratio in DAC miniDSP engine (1..15, or 0=16)
```

TSC[0:11h] - Reserved (xxh)

```
7-0   Reserved. Do not write to this register.
```

TSC[0:12h] - ADC NADC Value (01h)

```
7     ADC NADC divider is powered
        0: ADC NADC divider is powered down and ADC_DSP_CLK = DAC_DSP_CLK.
        1: ADC NADC divider is powered up.
6-0   ADC NADC divider  (1..127, or 0=128)
```

TSC[0:13h] - ADC MADC Value (01h)

```
7     ADC MADC divider is powered
        0: ADC MADC divider is powered down and ADC_MOD_CLK = DAC_MOD_CLK.
        1: ADC MADC divider is powered up.
6-0   ADC MADC divider  (1..127, or 0=128)
```

TSC[0:14h] - ADC AOSR Value (80h)

```
7-0   ADC OSR "AOSR" divider  (1..255, or 0=256)
```

AOSR should be an integral multiple of the decimation ratio in TSC[0:16h].

TSC[0:15h] - ADC IADC Value (80h)

```
7-0   Number of instruction for ADC miniDSP engine (IADC=N*2)
      (1..192 = 2..384 (N*2), or 0,193..255=Reserved)
```

IADC should be an integral multiple of the decimation ratio in TSC[0:16h].

TSC[0:16h] - ADC miniDSP Engine Decimation (04h)

```
7-4   Reserved
3-0   Decimation ratio in ADC miniDSP engine  (1..15, or 0=16)
```

TSC[0:17h,18h] - Reserved (xxh)

```
7-0   Reserved. Do not write to these registers.
```

TSC[0:19h] - CLKOUT MUX (00h)

```
7-3   Reserved
2-0   CDIV_CLKIN (0=MCLK, 1=BCLK, 2=SDIN, 3=PLL_CLK, 4=DAC_CLK(DSP),
      5=DAC_MOD_CLK, 6=ADC_CLK(DSP), 7=ADC_MOD_CLK)
```

TSC[0:1Ah] - CLKOUT Divider M Value (01h)

```
7     CLKOUT divider M Enable (0=Powered down, 1=Powered up)
6-0   CLKOUT divider M        (1..127, or 0=128)
```

DSi TSC[0:1Bh..23h], Codec Control

TSC[0:1Bh] - Codec Interface Control 1 (00h) (R/W)

```
7-6   Codec interface type        (0=I2S, 1=DSP, 2=RJF, 3=LJF)
5-4   Codec interface word length (0..3=16,20,24,32 bits)
3     BCLK Direction              (0=Input, 1=Output)
2     WCLK Direction              (0=Input, 1=Output)
```

```
  1     Reserved
  0     Driving SDOUT to High-Impedance for the Extra BCLK cycle when
         data is not being transferred (0=Disabled, 1=Enabled)
```

TSC[0:1Ch] - Data-Slot Offset Programmability (00h)

```
  7-0   Offset (0..255 = 0..255 BCLKs)
```

Note: Measured with respect to WCLK Rising Edge in DSP Mode.

TSC[0:1Dh] - Codec Interface Control 2 (00h)

```
  7-6   Reserved
  5     SDIN-to-SDOUT loopback (0=Disable, 1=Enable)
  4     ADC-to-DAC loopback    (0=Disable, 1=Enable)
  3     BCLK Invert            (0=No, 1=Invert)
  2     BCLK and WCLK active even with Codec powered down (0=No, 1=Yes)
  1-0   BDIV_CLKIN     (0=DAC_CLK, 1=DAC_MOD_CLK, 2=ADC_CLK, 3=ADC_MOD_CLK)
```

The BCLK settings in Bit2,3 do apply to both Primary and Secondary BCLK.

TSC[0:1Eh] - BCLK Divider N Value (01h)

```
  7     BCLK divider N Enable (0=Powered down, 1=Powered up)
  6-0   BCLK divider N        (1..127, or 0=128)
```

TSC[0:1Fh] - Codec Secondary Interface Control 1 (00h)

```
  7-5   Secondary BCLK is obtained from    ;\(0=GPIO1, 1=SCLK, 2=MISO, 3=SDOUT,
  4-2   Secondary WCLK is obtained from    ;/ 4=GPIO2, 5=GPI1, 6=GPI2, 7=GPI3)
  1-0   Secondary SDIN is obtained from (0=GPIO1, 1=SCLK, 2=GPIO2, 3=GPI1)
```

TSC[0:20h] - Codec Secondary Interface Control 2 (00h)

```
  7-5   ADC_WCLK is obtained from  (0=GPIO1, 1=SCLK, 2=MISO, 3=Reserved,
  4     Reserved                        4=GPIO2, 5=GPI1, 6=GPI2, 7=GPI3)
  3     Codec/ClockGen BCLK source (0=Primary BCLK, 1=Secondary BCLK)
  2     Codec WCLK source          (0=Primary WCLK, 1=Secondary WCLK)
  1     Codec ADC_WCLK source      (0=DAC_WCLK, 1=ADC_WCLK)
  0     Codec SDIN source          (0=Primary SDIN, 1=Secondary SDIN)
```

TSC[0:21h] - Codec Secondary Interface Control 3 (00h)

```
  7     Primary BCLK output   (0=Internally generated BCLK, 1=Secondary BCLK)
  6     Secondary BCLK output (0=Primary BCLK, 1=Internally generated BCLK)
  5-4   Primary WCLK output  (0=DAC_fS, 1=ADC_fS, 2=Secondary WCLK, 3=Reserved)
  3-2   Secondary WCLK output (0=Primary WCLK, 1=DAC_fS, 2=ADC_fS, 3=Reserved)
  1     Primary SDOUT         (0=SDOUT from codec, 1=Secondary SDIN)
  0     Secondary SDOUT       (0=Primary SDIN, 1=SDOUT from codec)
```

TSC[0:22h] - I2C Bus Condition (00h)

```
  7-6   Reserved. Write only the reset value to these bits.
  5     Accept I2C general-call address  (0=No/Ignore, 1=Yes/Accept)
  4-0   Reserved. Write only zeros to these bits.
```

TSC[0:23h] - Reserved (xxh)

```
  7-0   Reserved. Write only zeros to these bits.
```

DSi TSC[0:24h..32h], Status and Interrupt Flags

TSC[0:24h] - ADC Flag Register (0xh) (R)

```
  7     ADC PGA applied gain = programmed gain (0=Differs, 1=Equal) (R)
  6     ADC powered                  (0=Powered down, 1=Powered up) (R)
  5     AGC saturated        (0=No/inrange, 1=Yes/saturated to max) (R)
  4-0   Reserved. Write only zeros to these bits.
```

Note on D5(?): Sticky flag bIts. These are read-only bits. They are automatically cleared once they are read and are set only if the source trigger occurs again.

TSC[0:25h] - DAC Flag Register (00h) (R)

```
7     Left-channel DAC powered             (0=Powered down, 1=Powered up) (R)
6     Reserved. Write only zero to this bit.
5     Left Headphone HPL driver powered    (0=Powered down, 1=Powered up) (R)
4     Left-channel class-D driver powered  (0=Powered down, 1=Powered up) (R)
3     Right-channel DAC powered            (0=Powered down, 1=Powered up) (R)
2     Reserved. Write only zero to this bit.
1     Right Headphone HPR driver powered   (0=Powered down, 1=Powered up) (R)
0     Right-channel class-D driver powered (0=Powered down, 1=Powered up) (R)
```

TSC[0:26h] - DAC Flag Register (00h) (R)

```
7-5   Reserved. Do not write to these bits.
4     Left-channel DAC PGA applied gain=programmed gain  (0=Differs, 1=Equal)
3-1   Reserved. Write only zeros to these bits.
0     Right-channel DAC PGA applied gain=programmed gain (0=Differs, 1=Equal)
```

TSC[0:27h] - Overflow Flags (00h) (R)

```
7     Left-Channel DAC Overflow Flag           (0=None, 1=Overflow) (R)
6     Right-Channel DAC Overflow Flag          (0=None, 1=Overflow) (R)
5     DAC Barrel Shifter Output Overflow Flag  (0=None, 1=Overflow) (R)
4     Reserved. Write only zeros to these bits.
3     Delta-Sigma Mono ADC Overflow Flag       (0=None, 1=Overflow) (R)
2     Reserved. Write only zero to this bit.
1     ADC Barrel Shifter Output Overflow Flag  (0=None, 1=Overflow) (R)
0     Reserved. Write only zero to this bit.
```

Sticky flag bIts. These are read-only bits. They are automatically cleared once they are read and are set only if the source trigger occurs again.

TSC[0:28h..2Bh] - Reserved (xxh)

```
7-0   Reserved. Write only the reset value to these bits.
```

TSC[0:2Ch] - Interrupt Flags DAC, sticky (00h..30h) (R)

```
7     Short-circuit detected at HPL/left class-D driver         (0=No, 1=Yes)
6     Short-circuit detected at HPR/right class-D driver        (0=No, 1=Yes)
5     Headset button pressed                                    (0=No, 1=Yes)
4     Headset insertion/removal is detected                     (0=No, 1=Yes)
3     Left DAC signal power vs signal threshold of DRC (0=Less/Equal,1=Above)
2     Right DAC signal power vs signal threshold of DRC(0=Less/Equal,1=Above)
1     DAC miniDSP Engine Standard Interrupt-Port Output  (0=Read 0, 1=Read 1)
0     DAC miniDSP Engine Auxiliary Interrupt-Port Output (0=Read 0, 1=Read 1)
```

Sticky flag bIts. These are read-only bits. They are automatically cleared once they are read and are set only if the source trigger occurs again.

TSC[0:2Dh] - Interrupt Flags ADC, sticky (00h..18h) (R)

```
7     Reserved. Write only zero to this bit.
6     ADC signal power vs noise threshold for AGC (0=Greater, 1=Less)
5     Reserved. Write only zeros to these bits.
4     ADC miniDSP Engine Standard Interrupt Port Output  (0=Read 0, 1=Read 1)
3     ADC miniDSP Engine Auxiliary Interrupt Port Output (0=Read 0, 1=Read 1)
2     DC measurement using Delta Sigma Audio ADC
        (0=Not available, 1=Not available, too, uh?)
1-0   Reserved. Write only zeros to these bits.
```

Sticky flag bIts. These are read-only bits. They are automatically cleared once they are read and are set only if the source trigger occurs again.

TSC[0:2Eh] - Interrupt Flags DAC, non-sticky? (00h..30h) (R)

```
7     Short circuit detected at HPL/left class-D driver         (0=No, 1=Yes)
```

```
  6     Short circuit detected at HPR/right class-D driver        (0=No, 1=Yes)
  5     Headset button pressed                                    (0=No, 1=Yes)
  4     Headset removal/insertion detected             (0=Removal, 1=Insertion)
  3     Left DAC signal power vs signal threshold of DRC     (0=Below, 1=Above)
  2     Right DAC signal power vs signal threshold of DRC    (0=Below, 1=Above)
  1     DAC miniDSP Engine Standard Interrupt Port Output  (0=Read 0, 1=Read 1)
  0     DAC miniDSP Engine Auxiliary Interrupt Port Output (0=Read 0, 1=Read 1)
```
This is (almost?) same as TSC[0:2Ch]. Maybe this is current state (non-sticky)?

TSC[0:2Fh] - Interrupt Flags ADC, non-sticky? (00h..18h) (R)
```
  7     Reserved
  6     Delta-sigma mono ADC signal power vs noise threshold for left AGC
  5     Reserved                                          (0=Greater, 1=Less)
  4     ADC miniDSP Engine Standard Interrupt Port Output  (0=Read 0, 1=Read 1)
  3     ADC miniDSP Engine Auxiliary Interrupt Port Output (0=Read 0, 1=Read 1)
  2     DC measurement using Delta Sigma Audio ADC
          (0=Not available, 1=Not available, too, uh?)
  1-0   Reserved. Write only zeros to these bits.
```
This is (almost?) same as TSC[0:2Dh]. Maybe this is current state (non-sticky)?

TSC[0:30h] - INT1 Control Register (Select INT1 Sources) (00h)
TSC[0:31h] - INT2 Control Register (Select INT2 Sources) (00h)
```
  7     Headset-insertion detect                                  (0=Off, 1=On)
  6     Button-press detect                                       (0=Off, 1=On)
  5     DAC DRC signal-power                                      (0=Off, 1=On)
  4     ADC AGC noise                                             (0=Off, 1=On)
  3     Short-circuit                                             (0=Off, 1=On)
  2     Engine-generated                                          (0=Off, 1=On)
  1     DC measurement using Delta Sigma Audio ADC data-available (0=Off, 1=On)
  0     INT duration (0=Pulse Once, 1=Pulse Repeatedly until Acknowledge)
```
Bit1-7 select which sources shall trigger INT1/INT2, a few more sources can be selected in TSC[0:32h].Bit5-7. Bit0 selects how the INT1/INT2 signal shall be pulsed (once with 2ms length, or repeating every 4ms with 50% duty, until it gets acknowledged by reading TSC[0:2Ch,2Dh,32h]).

TSC[0:32h] - INT1 and INT2 Control Register (00h)
```
  7     INT1 upon SAR measurement data-out-of-threshold range   (0=Off, 1=Off?)
  6     INT1 upon Pen touch/SAR data-available                    (0=Off, 1=On)
  5     INT2 upon SAR measurement data-out-of-threshold range   (0=Off, 1=Off?)
  4     Reserved
  3     Pen touch detected                          (0=No, 1=Touch)     (R)
  2     Data available for read                     (0=No, 1=Available) (R)
  1     SAR data out of programmed threshold range  (0=No, 1=Out)       (R)
  0     Reserved. Write only the default value to this bit.             (R)
```

DSi TSC[0:33h..3Bh], Pin Control

TSC[0:33h] - GPIO1 In/Out Pin Control (00h..C2h)
TSC[0:34h] - GPIO2 In/Out Pin Control (00h..C2h)
```
  7-6   Reserved. Do not write any value other than reset value.
  5-2   GPIOx Mode                                                  (R/W)
          0 = GPIOx disabled (input and output buffers powered down)
          1 = GPIOx input mode (as secondary BCLK/WCLK/SDIN input, or
                as ADC_WCLK input, Dig_Mic_In or in ClockGen block)
          2 = GPIOx input mode (as GPI general-purpose input)
          3 = GPIOx output = general-purpose output
          4 = GPIOx output = CLKOUT output
          5 = GPIOx output = INT1 output
          6 = GPIOx output = INT2 output
          7 = GPIOx output = ADC_WCLK output for codec interface
          8 = GPIOx output = secondary BCLK output for codec interface
```

```
        9 = GPIOx output = secondary WCLK output for codec interface
        10 = GPIOx output = ADC_MOD_CLK output for the digital microphone
        11 = GPIOx output = secondary SDOUT for codec interface
        12 = GPIOx output = TouchScreen/SAR ADC interrupt (active-low),
        13-15 = Reserved                              as PINTDAV signal
1     GPIOx input buffer value (0 or 1)                          (R)
0     GPIOx general-purpose output value (0 or 1)                (R/W)
```

TSC[0:35h] - SDOUT (OUT Pin) Control (12h)

```
7-5   Reserved
4     SDOUT bus keeper (0=Enabled, 1=Disabled)
3-1   SDOUT Mode
        0 = SDOUT disabled (output buffer powered down)
        1 = SDOUT = primary SDOUT output for codec interface
        2 = SDOUT = general-purpose output
        3 = SDOUT = CLKOUT output
        4 = SDOUT = INT1 output
        5 = SDOUT = INT2 output
        6 = SDOUT = secondary BCLK output for codec interface
        7 = SDOUT = secondary WCLK output for codec interface
0     SDOUT general-purpose output value (0 or 1)
```

TSC[0:36h] - SDIN (IN Pin) Control (02h or 03h)

```
7-3   Reserved
2-1   SDIN Mode
        0 = SDIN disabled (input buffer powered down)
        1 = SDIN enabled (as codec SDIN, Dig_Mic_In, or in ClockGen block)
        2 = SDIN enabled (as GPI general-purpose input)
        3 = Reserved
0     SDIN input-buffer value (0 or 1)                           (R)
```

TSC[0:37h] - MISO (OUT Pin) Control (02h)

```
7-5   Reserved
4-1   MISO Mode
        0 = MISO disabled (output buffer powered down)
        1 = MISO = MISO output for SPI interface (or disabled for I2C)
        2 = General-purpose output
        3 = MISO = CLKOUT output
        4 = MISO = INT1 output
        5 = MISO = INT2 output
        6 = MISO = ADC_WCLK output for codec interface
        7 = MISO = ADC_MOD_CLK output for the digital microphone
        8 = MISO = secondary SDOUT for codec interface
        9 = MISO = secondary BCLK output for codec interface
        10 = MISO = secondary WCLK output for codec interface
        11-15 = Reserved
0     MISO general-purpose output value (0 or 1)
```

TSC[0:38h] - SCLK (IN Pin) Control (02h..03h)

```
7-3   Reserved
2-1   SCLK Mode
        0 = SCLK disabled (input buffer powered down)
        1 = SCLK enabled (for the SPI interface)
        2 = SCLK enabled (as a GPI general-purpose input)
        3 = SCLK enabled (as secondary SDIN/BCLK/WCLK input,
                          or as ADC_WCLK input, or Dig_Mic_In)
0     SCLK input buffer value (0 or 1)                           (R)
```

TSC[0:39h] - GPI1 and GPI2 Pin Control (00h..11h)

```
7     Reserved. Write only zero to this bit.
6-5   GPI1 Mode
        0 = GPI1 disabled (input buffer powered down)
        1 = GPI1 enabled (as secondary SDIN/BCLK/WCLK input, or ADC_WCLK inp)
```

```
        2 = GPI1 enabled (as a GPI general-purpose input)
        3 = Reserved (unlike below GPI2)
4     GPI1 pin value (0 or 1)         (R)
3     Reserved. Write only zero to this bit.
2-1   GPI2 Mode
        0 = GPI2 disabled (input buffer powered down)
        1 = GPI2 enabled (as secondary BCLK/WCLK input, or ADC_WCLK input)
        2 = GPI2 enabled (as a GPI general-purpose input)
        3 = GPI2 enabled (as an HP_SP input)
0     GPI2 pin value (0 or 1)         (R)
```

TSC[0:3Ah] - GPI3 Pin Control (00h..10h)

```
7     Reserved. Write only zero to this bit.
6-5   GPI3 Mode
        0 = GPI3 disabled (input buffer powered down)
        1 = GPI3 enabled (as secondary BCLK/WCLK input, or ADC_WCLK input)
        2 = GPI3 enabled (as a GPI general purpose input)
        3 = Reserved (Undocumented - used by DSi?)
4     GPI3 pin value (0 or 1)         (R)
3-0   Reserved. Write only zeros to these bits.
```

TSC[0:3Bh] - Reserved (xxh)

```
7-0   Reserved. Write only zeros to these bits.
```

DSi TSC[0:3Ch..55h], DAC/ADC and Beep

TSC[0:3Ch] - DAC Instruction Set (01h)

```
7-5   Reserved. Write only default value.
4-0   DAC Signal Processing Block
        0      = DAC miniDSP is used for signal processing
        1..25  = DAC Signal Processing Block PRB_P1 .. PRB_P25
        26..31 = Reserved. Do not use.
```

TSC[0:3Dh] - ADC Instruction Set (04h)

```
7-5   Reserved. Write only default values.
4-0   ADC Signal Processing Block
        0      = ADC miniDSP is used for signal processing
        1..3   = Reserved
        4..6   = ADC Signal Processing Block PRB_R4 .. PRB_R6
        7..9   = Reserved
        10..12 = ADC Signal Processing Block PRB_R10 .. PRB_R12
        13..15 = Reserved
        16..18 = ADC Signal Processing Block PRB_R16 .. PRB_R18
        19..31 = Reserved. Do not write these sequences to these bits.
```

TSC[0:3Eh] - Programmable Instruction Mode-Control Bits (00h)

```
7     Reserved
6     ADC miniDSP Engine Auxiliary Control bit A (0 or 1)
5     ADC miniDSP Engine Auxiliary Control bit B (0 or 1)
4     Reset ADC miniDSP instruction counter at start of new frame (0=Yes)
3     Reserved
2     DAC miniDSP Engine Auxiliary Control bit A (0 or 1)
1     DAC miniDSP Engine Auxiliary Control bit B (0 or 1)
0     Reset DAC miniDSP instruction counter at start of new frame (0=Yes)
```

Above DAC/ADC bit A and B can be used for conditional instructions like JMP.

TSC[0:3Fh] - DAC Data-Path Setup (14h)

```
7     Left-channel DAC            (0=Powered down, 1=Powered up)
6     Right-channel DAC           (0=Powered down, 1=Powered up)
5-4   Left-channel DAC data path  (0=Off, 1=Left Data, 2=Right Data, 3=Both)
```

```
  3-2   Right-channel DAC data path (0=Off, 1=Right Data, 2=Left Data, 3=Both)
  1-0   DAC channel volume control soft-stepping (0=One step per sample,
         1=One step per 2 samples, 2=Disabled, 3=Reserved)
```
Wheras, Both=((L+R)/2).

TSC[0:40h] - DAC Volume Control (0Ch)
```
  7-4   Reserved. Write only zeros to these bits.
  3     Left-channel DAC  (0=Not muted, 1=Muted)
  2     Right-channel DAC (0=Not muted, 1=Muted)
  1-0   DAC Mono/Stereo Volume
          0: Use Left/Right volume control for Left/Right channels ("stereo")
          1: Use Right volume control for Both channels            ("mono")
          2: Use Left volume control for Both channels             ("mono")
          3: Same as 0                                             ("stereo")
```

TSC[0:41h] - DAC Left Volume Control (00h)
TSC[0:42h] - DAC Right Volume Control (00h)
```
  7-0  Digital gain in 0.5dB units (-127..+48 = -63.5dB..+24dB, Other=Reserved)
```

TSC[0:43h] - Headset Detection (00h..60h)
```
  7     Headset detection Enable (0=Disabled, 1=Enabled)
  6-5   Headset detection (0=None, 1=Headset, 2=Reserved, 3=Headset+Mic) (R)
  4-2   Debounce for Glitch Rejection During Headset Detection
          (0..5 = 16ms, 32ms, 64ms, 128ms, 256ms, 512ms, 6..7=Reserved)
          (when TSC[3:10h] set to 1MHz)
  1-0   Debounce for Glitch Rejection During Headset Button-Press Detection
          (0..3 = 0ms, 8ms, 16ms, 32ms) (when TSC[3:10h] set to 1MHz)
```
Sampling is 8x faster than above timings (eg. time=32ms uses 4ms sampling).

TSC[0:44h] - DRC Control 1 (0Fh)
```
  7     Reserved. Write only the reset value to these bits.
  6     DRC for left channel  (0=Disabled, 1=Enabled)
  5     DRC for right channel (0=Disabled, 1=Enabled)
  4-2   DRC threshold  (0..7 = -3dB,-6dB,-9dB,-12dB,-15dB,-18dB,-21dB,-24dB)
  1-0   DRC hysteresis (0..3 = +0dB,+1dB,+2dB,+3dB)
```

TSC[0:45h] - DRC Control 2 (38h)
```
  7     Reserved. Write only the reset value to these bits.
  6-3   DRC Hold Time
          0  = DRC Hold Disabled        ;-disable
          1  = 32 DAC Word Clocks       ;\
          2  = 64 DAC Word Clocks       ;
          3  = 128 DAC Word Clocks      ;
          4  = 256 DAC Word Clocks      ; powers of 2
          5  = 512 DAC Word Clocks      ;
          6  = 1024 DAC Word Clocks     ;
          7  = 2048 DAC Word Clocks     ;
          8  = 4096 DAC Word Clocks     ;
          9  = 8192 DAC Word Clocks     ;
          10 = 16384 DAC Word Clocks    ;/
          11 = 1*32768 DAC Word Clocks  ;\
          12 = 2*32768 DAC Word Clocks  ;
          13 = 3*32768 DAC Word Clocks  ; multiples of 32768
          14 = 4*32768 DAC Word Clocks  ;
          15 = 5*32768 DAC Word Clocks  ;/
  2-0   Reserved. Write only the reset value to these bits.
```

TSC[0:46h] - DRC Control 3 (00h)
```
  7-4   DRC attack rate, "(4 SHR N) dB per DAC Word Clock"
        (0=4dB, 1=2dB, 2=1dB, ..., 15=0.000122dB per DAC Word Clock)
  3-0   DRC decay rate, "(1 SHR (N+6)) dB per DAC Word Clock"
        (0=0.0156dB, 1=0.00781dB, ..., 15=0.000000476dB per DAC Word Clock)
```

TSC[0:47h] - Beep Generator and Left Beep Volume (00h)

```
7     Beep Generator Enable (0=Disabled/Duration ended, 1=Enabled/Busy)
        (self-clearing based on beep duration)
6     Auto beep generator on pen touch (0=Disabled, 1=Enabled)
        (CODEC_CLKIN should be available for this and is
        used whenever touch is detected).
5-0   Left-channel beep volume control "(2-N)dB" (0..63 = +2dB .. -61dB)
```

The beep generator is only available in PRB_P25 DAC processing mode.

TSC[0:48h] - Beep Generator and Right Beep Volume (00h)

```
7-6   Beep Mono/Stereo Volume
        0: Use Left/Right volume control for Left/Right channels ("stereo")
        1: Use Right volume control for Both channels            ("mono")
        2: Use Left volume control for Both channels             ("mono")
        3: Same as 0                                             ("stereo")
5-0   Right-channel beep volume control "(2-N)dB" (0..63 = +2dB .. -61dB)
```

The beep generator is only available in PRB_P25 DAC processing mode.

TSC[0:49h,4Ah,4Bh] - Beep Length MSB,MID,LSB (0000EEh)

```
23-0  Number of samples for which beep need to be generated (24bit)
```

TSC[0:4Ch,4Dh] - Beep Frequency Sin(x) MSB,LSB (10D8h)
TSC[0:4Eh,4Fh] - Beep Frequency Cos(x) MSB,LSB (7EE3h)

```
15-0  Beep Frequency sin/cos values (16bit, each)
```

These registers should be set to sin(2pi*fin/fS) and cos(2pi*fin/fS) accordingly; where fin is the beep frequency and fS is the DAC sample rate.

TSC[0:50h] - Reserved (xxh)

```
7-0   Reserved. Write only the reset value to these bits.
```

TSC[0:51h] - ADC Digital Mic (00h)

```
7     ADC channel   (0=Powered Down, 1=Powered Up)
6     Reserved
5-4   Digital microphone input  (0=GPIO1, 1=SCLK, 2=SDIN, 3=GPIO2)
3     Digital microphone for delta-sigma mono ADC channel (0=Off, 1=On)
2     Reserved
1-0   ADC channel volume control soft-stepping (0=One step per sample,
       1=One step per 2 samples, 2=Disabled, 3=Reserved)
```

TSC[0:52h] - ADC Digital Volume Control Fine Adjust (80h)

```
7     ADC channel   (0=Not muted, 1=Muted)
6-4   Delta-Sigma Mono ADC Channel Volume Control Fine Gain
      (0=0dB, 1=-0.1dB, 2=-0.2dB, 3=-0.3dB, 4=-0.4dB, 5..7=Reserved)
3-0   Reserved. Write only zeros to these bits.
```

TSC[0:53h] - ADC Digital Volume Control Coarse Adjust (00h)

```
7     Reserved
6-0   Delta-Sigma Mono ADC Channel Volume Control Coarse Gain
        0..39    = Reserved
        40       = -12 dB
        39       = -11.5 dB
        ...
        103      = +19.5 dB
        104      = +20 dB
        105..127 = Reserved
```

TSC[0:54h,55h] - Reserved (xxh)

```
7-0   Reserved. Write only the reset value to these bits.
```

DSi TSC[0:56h..7Fh], AGC and ADC

Auto-Gain-Control & Analog-to-Digital Converter

TSC[0:56h] - AGC Control 1 (00h)
```
  7     AGC (0=Disabled, 1=Enabled)
  6-4   AGC target level     (0=-5.5dB, 1=-8dB, 2=-10dB, 3=-12dB,
                              4=-14dB, 5=-17dB, 6=-20dB, 7=-24dB)
  3-0   Reserved. Write only zeros to these bits.
```

TSC[0:57h] - AGC Control 2 (00h)
```
  7-6   AGC hysterysis setting (0=1dB, 1=2dB, 2=4dB, 3=Disable AGC hysterysis)
  5-1   AGC noise threshold (and silence detection)
          0   = AGC noise/silence detection is disabled.
          1   = AGC noise threshold = -30dB
          2   = AGC noise threshold = -32dB
          3   = AGC noise threshold = -34dB
          ...
          29  = AGC noise threshold = -86dB
          30  = AGC noise threshold = -88dB
          31  = AGC noise threshold = -90dB
  0     Reserved. Write only zero to this bit.
```

TSC[0:58h] - AGC Maximum Gain (7Fh, uh that's 7Fh=Reserved?)
```
  7     Reserved. Write only zero to this bit.
  6-0   AGC maximum gain in 0.5dB units (0..119=0..+59.5dB, 120..127=Reserved)
```

TSC[0:59h] - AGC Attack Time (00h)
TSC[0:5Ah] - AGC Decay Time (00h)
```
  7-3   AGC attack/decay time, (N*2+1)*32/fS (0..31 = 1*32/fS .. 63*32/fS)
  2-0   AGC attack/decay time Multiply factor, 1 SHL N (0..7 = 1..128)
```
Whereas, fS is the ADC sample rate.

TSC[0:5Bh] - AGC Noise Debounce (00h)
```
  7-5   Reserved. Write only zeros to these bits.
  4-0   AGC noise debounce
          0..5   = 0/fS, 4/fS, 8/fS, 16/fS, 32/fS, 64/fS     ;\powers of 2
          6..10  = 128/fS, 256/fS, 512/fS, 1024/fS, 2048/fS  ;/
          11..14 = 1*4096/fS, 2*4096/fS, 3*4096/fS           ;\multiples
          14..31 = 4*4096/fS, .., 20*4096/fS, 21*4096/fS     ;/of 4096
```

TSC[0:5Ch] - AGC Signal Debounce (00h)
```
  7-4   Reserved. Write only zeros to these bits.
  3-0   AGC signal debounce
          0..5   = 0/fS, 4/fS, 8/fS, 16/fS, 32/fS, 64/fS     ;\powers of 2
          6..9   = 128/fS, 256/fS, 512/fS, 1024/fS           ;/
          10..13 = 1*2048/fS, 2*2048/fS, 3*2048/fS           ;\multiples
          13..15 = 4*2048/fS, 5*2048/fS, 6*2048/fS           ;/of 2048
```

TSC[0:5Dh] - AGC Gain-Applied Reading (xxh) (R)
```
  7-0   Gain applied by AGC in 0.5dB units (-24..+119 = -12dB..+59.5dB)   (R)
```

TSC[0:5Eh...65h] - Reserved (xxh)
```
  7-0   Reserved. Do not write to these registers.
```

TSC[0:66h] - ADC DC Measurement 1 (00h)
```
  7     DC measurement for mono ADC channel (0=Disabled, 1=Enabled)
```

```
6     Reserved. Write only reset value.
5     DC measurement is done based on
        0: 1st order sinc filter with averaging of 2^D.
        1: 1st order low-pass IIR filter whose coefficients
            are calculated based on D value.
4-0   DC Meaurement D setting (1..20 = D=1 .. D=20) (0 or 21..31=Reserved)
```

TSC[0:67h] - ADC DC Measurement 2 (00h)

```
7     Reserved. Write only reset value.
6     DC measurement data update (0=Enabled, 1=Disabled/allow stable reading)
      (Disabled: user can read the last updated data without corruption)
5     For IIR based DC measurement, the measurment value is
        0: the instantaneous output of the IIR filter
        1: update before periodic clearing of the IIR filter
4-0   IIR based DC measurment, average time setting:
        0       Infinite average is used
        1       Averaging time is 2^1 ADC modulator clock periods
        2       Averaging time is 2^2 ADC modulator clock periods
        ...
        19      Averaging time is 2^19 ADC modulator clock periods
        20      Averaging time is 2^20 ADC modulator clock periods
        21..31  Reserved. Don't use.
```

TSC[0:68h,69h,6Ah] - ADC DC Measurement Output MSB,MID,LSB (R) (000000h)

```
23-0  ADC DC Measurement Output (24bit)
```

TSC[0:6Bh...73h] - Reserved (xxh)

```
7-0   Reserved. Do not write to these registers.
```

TSC[0:74h] - VOL/MICDET-Pin SAR ADC - Volume Control (00h)

```
7     DAC volume control is controlled by,
        0: controlled by control register (7-bit Vol ADC is powered down)
        1: controlled by pin (analog volume input)
6     Clock for the 7-bit Vol ADC for pin volume control,
        0: Internal on-chip RC oscillator
        1: External MCLK
5-4   Hysteresis
        0: No hysteresis for volume control ADC output
        1: Hysteresis of +/-1 bit
        2: Hysteresis of +/-2 bits
        3: Reserved. Do not write this sequence to these bits.
3     Reserved. Write only reset value.
2-0   Throughput of the 7-bit Vol ADC for pin volume control,
      When Bit6=1 and external MCLK is 12MHz:
        (0..7=15.625Hz, 31.25Hz, 62.5Hz, 125Hz, 250Hz, 500Hz, 1000Hz, 2000Hz)
      When Bit6=0 (use Internal oscillator):
        (0..7=10.68Hz, 21.35Hz, 42.71Hz, 85Hz?, 170Hz, 340Hz, 680Hz, 1370Hz)
```

TSC[0:75h] - VOL/MICDET-Pin Gain (xxh) (R)

```
7     Reserved. Write only zero to this bit.
6-0   Gain applied by pin volume control
        0   = +18 dB
        1   = +17.5 dB
        2   = +17 dB
        ...
        35  = +0.5 dB
        36  = 0 dB
        37  = -0.5 dB
        ...
        89  = -26.5 dB
        90  = -27 dB    ;below in 1dB steps instead of 0.5dB steps !
        91  = -28 dB
        ...
```

```
          125 = -62 dB
          126 = -63 dB
          127 = Reserved
```

TSC[0:76h...7Fh] - Reserved (xxh)

```
  7-0   Reserved. Do not write to these registers.
```

DSi TSC[1:xxh], DAC and ADC Routing, PGA, Power-Controls and MISC Logic

TSC[1:01h..1Dh] - Reserved (xxh)

```
  7-0  Reserved. Do not write to these registers.
```

TSC[1:1Eh] - Headphone and Speaker Amplifier Error Control (00h)

```
  7-2  Reserved
  1    Reset HPL/HPR power-up bits upon short-circuit detect (0=Yes, 1=No)
  0    Reset SPL/SPR power-up bits upon short-circuit detect (0=Yes, 1=No)
```

The HPL/HPR auto-reset occurs only if TSC[1:1Fh].Bit1=1 (action=power down).

TSC[1:1Fh] - Headphone Drivers (04h)

```
  7    HPL output driver (0=Powered down, 1=Powered up)
  6    HPR output driver (0=Powered down, 1=Powered up)
  5    Reserved. Write only zero to this bit.
  4-3  Output common-mode voltage (0=1.35V, 1=1.5V, 2=1.65V, 3=1.8V)
  2    Reserved. Write only 1 to this bit. (!!!)
  1    Action when short-circuit protection is enabled/detected,
           0=Limit the maximum current to the load.
           1=Power down the output driver.
  0    Short-circuit detected on the headphone driver (0=No, 1=Yes) (R)
```

TSC[1:20h] - Class-D Speaker Amplifier (06h)

```
  7    Left-channel class-D output driver  (0=Powered down, 1=Powered up)
  6    Right-channel class-D output driver (0=Powered down, 1=Powered up)
  5-1  Reserved. Write only the reset value (00011b) to these bits (!!!)
  0    Short-circuit is detected on the class-D driver (0=No, 1=Yes) (R)
```

Bit0 is Valid only if class-D amplifier is powered up. For short-circuit flag sticky bit, see TSC[0:2Ch].

TSC[1:21h] - HP Output Drivers POP Removal Settings (3Eh)

```
  7    If power down sequence is activated by device software power down
       using TSC[1:2Eh].Bit7 then power down DAC,
         0: simultaneously with the HP and SP amplifiers.
         1: after HP and SP amplifiers are completely powered down.
         (the latter setting is to optimize power-down POP).
  6-3  Driver power-on time (at 8.2MHz) (1=15.3us, 2=153us, 3=1.53ms,
         4=15.3ms,5=76.2ms, 6=153ms, 7=304ms, 8=610ms, 9=1.22s, 10=3.04s,
         11=6.1s, 12..15=Reserved)
  2-1  Driver ramp-up step time (8.2MHz) (0=0ms, 1=0.98ms, 2=1.95ms, 3=3.9ms)
  0    Weakly driven output common-mode voltage is generated from,
         0=resistor divider of the AVDD supply.
         1=band-gap reference.
```

TSC[1:22h] - Output Driver PGA Ramp-Down Period Control (00h)

```
  7    Reserved. Write only the reset value to this bit.    (USED on DSi!)
  6-4  Speaker Power-Up Wait Time (at 8.2MHz) (0=0 ms, 1=3.04 ms, 2=7.62 ms,
         3=12.2 ms, 4=15.3 ms, 5=19.8 ms, 6=24.4 ms, 7=30.5 ms)
  3-0  Reserved. Write only the reset value to these bits.
```

TSC[1:23h] - DAC_L and DAC_R Output Mixer Routing (00h)

```
  7-6  DAC_L route      (0=Nowhere, 1=To L-Mixer, 2=Direct to HPL, 3=Reserved)
  5    MIC input routed to the left-channel mixer amplifier      (0=No, 1=Yes)
  4    AUX1 input routed to the left-channel mixer amplifier     (0=No, 1=Yes)
  3-2  DAC_R route      (0=Nowhere, 1=To R-Mixer, 2=Direct to HPR, 3=Reserved)
  1    AUX1 input routed to the right-channel mixer amplifier    (0=No, 1=Yes)
  0    HPL driver output routed to HPR driver (for differential) (0=No, 1=Yes)
```

TSC[1:24h] - Analog Volume to HPL (Left Headphone) (7Fh)
TSC[1:25h] - Analog Volume to HPR (Right Headphone) (7Fh)
TSC[1:26h] - Analog Volume to SPL (Left Speaker) (7Fh)
TSC[1:27h] - Analog Volume to SPR (Right Speaker) (7Fh)

```
  7    Analog volume control routed to HPx/SPx output driver (0=No, 1=Yes)
  6-0  Analog volume control gain (non-linear) (0 dB to -78 dB)
```

See Table 5-37 and Table 5-38, uh?

TSC[1:28h] - HPL Driver (Left Headphone) (02h)
TSC[1:29h] - HPR Driver (Right Headphone) (02h)

```
  7    Reserved. Write only zero to this bit.
  6-3  HPx driver PGA  (0..9 = 0dB..9dB, 10..15=Reserved)
  2    HPx driver      (0=Muted, 1=Not muted)
  1    HPx driver during power down (0=Weakly driven to a common mode,
        1=High-impedance)
  0    All programmed gains to HPx have been applied (0=Not yet, 1=Yes/all) (R)
```

TSC[1:2Ah] - SPL Driver (Left Speaker) (00h)
TSC[1:2Bh] - SPR Driver (Right Speaker) (00h)

```
  7-5  Reserved. Write only zeros to these bits.
  4-3  SPx class-D driver output stage gain (0=6dB, 1=12dB, 2=18dB, 3=24dB)
  2    SPx class-D driver (0=Muted, 1=Not muted)
  1    Reserved. Write only zero to this bit.
  0    All programmed gains to SPx have been applied (0=Not yet, 1=Yes/all) (R)
```

TSC[1:2Ch] - HP Driver Control (00h)

```
  7-5  Debounce time for the headset short-circuit detection
         (0..7 = 0us, 8us, 16us, 32us, 64us, 128us, 256us)
         (when TSC[3:10h] set to 1MHz)
  4-3  DAC Performance (0=Normal, 1=Increased, 2=Reserved, 3=Further Increased)
       (increased: by increased current, further: by increased current gain)
  2    HPL output driver type (0=Headphone, 1=Lineout)
  1    HPR output driver type (0=Headphone, 1=Lineout)
  0    Reserved. Write only zero to this bit.
```

The clock used for the debounce has a clock period = debounce duration/8.

TSC[1:2Dh] - Reserved (xxh)

```
  7-0  Reserved. Do not write to these registers.
```

TSC[1:2Eh] - MICBIAS (00h)

```
  7    Device software power-down (0=Disabled, 1=PowerDown?-Enabled)
  6-4  Reserved. Write only zeros to these bits.
  3    Power up programmed MICBIAS  (0=Only if Headset inserted, 1=Always)
  2    Reserved. Write only zero to this bit.
  1-0  MICBIAS output (0=Off, 1=2V, 2=2.5V, 3=AVDD)
```

TSC[1:2Fh] - MIC PGA (80h)

```
  7    MIC PGA            (0=Controlled by bits6-0, 1=Force 0dB)
  6-0  PGA in 0.5dB units (0..119 = 0..59.5dB, 120..127=Reserved)
```

TSC[1:30h] - P-Terminal Delta-Sigma Mono ADC Channel Fine-Gain Input (00h)

```
  7-6  MIC to MIC PGA feed-forward    (0=Off, 1=10kOhm, 2=20kOhm, 3=40kOhm)
  5-4  AUX1 to MIC PGA feed-forward   (0=Off, 1=10kOhm, 2=20kOhm, 3=40kOhm)
```

```
  3-2  AUX2 to MIC PGA feed-forward   (0=Off, 1=10kOhm, 2=20kOhm, 3=40kOhm)
  1-0  Reserved. Write only zeros to these bits.
```
Program Bit7-6 of registers TSC[1:30h] and TSC[1:31h] with same value. Input impedance selection affects the microphone PGA gain. See the Analog Front End section for details.

TSC[1:31h] - M-Terminal ADC Input Selection (00h)
```
  7-6  CM to MIC PGA feed-forward     (0=Off, 1=10kOhm, 2=20kOhm, 3=40kOhm)
  5-4  AUX2 to MIC PGA feed-forward   (0=Off, 1=10kOhm, 2=20kOhm, 3=40kOhm)
  3-0  Reserved. Write only zeros to these bits.
```
Program Bit7-6 of registers TSC[1:30h] and TSC[1:31h] with same value. Input impedance selection affects the microphone PGA gain. See the Analog Front End section for details.

TSC[1:32h] - Input CM Settings (00h)
```
  7    MIC input  (0=Floating, 1=Connected to CM internally)
        (when not used for MIC PGA and analog bypass)
  6    AUX1 input (0=Floating, 1=Connected to CM internally)
        (when not used for MIC PGA and analog bypass)
  5    AUX2 input (0=Floating, 1=Connected to CM internally)
        (when not used for MIC PGA)
  4-1  Reserved. Write only zeros to these bits.
  0    All programmed gains to ADC have been applied (0=Not yet, 1=Yes/all) (R)
```

TSC[1:33h..FFh] - Reserved (xxh)
```
  7-0  Reserved. Write only the reset value to these bits.
```

DSi TSC[3:xxh], Touchscreen/SAR Control and TSC[FCh:xxh], Buffer

TSC[3:01h] - Reserved (xxh)
```
  7-0   Reserved. Write only the reset value to these bits.
```

TSC[3:02h] - SAR ADC Control 1 (00h)
```
  7     Stop  (0=Normal mode, 1=Stop conversion and power down SAR ADC)
  6-5   SAR ADC resolution   (0=12bit, 1=8bit, 2=10bit, 3=12bit)
  4-3   SAR ADC clock divider
          0 = Div1 (use for 8bit resolution mode only) (This divider is only
                    for the conversion clock generation, not for other logic)
          1 = Div2 (use for 8bit/10bit resolution mode only)
          2 = Div4 (recommended for better performance in 8bit/10bit mode)
          3 = Div8 (recommended for better performance in 12bit mode)
          (See Figure 5-40, uh?)
  2     Filter used for on-chip data averaging (0=Mean, 1=Median) (if enabled)
  1-0   On-chip data averaging for mean/median filter
          0 = On-chip data averaging disabled
          1 = 4-data averaging (mean),  or 5-data averaging (median)
          2 = 8-data averaging (mean),  or 9-data averaging (median)
          3 = 16-data averaging (mean), or 15-data averaging (median)
```

TSC[3:03h] - SAR ADC Control 2 (00h)
```
  7     Conversions controlled,
          0 = Host-controlled conversions
          1 = Self-controlled conversions for touch screen based on pen touch
  6     Reserved. Write only zero to this bit.
  5-2   Conversion Mode
          0 = No scan
          1 = Scan X/Y         ;\Even in host-controlled mode ;\until either
          2 = Scan X/Y/Z1/Z2   ;/                             ; pen is lifted,
          3 = Scan X           ;\                             ; or a stop bit
          4 = Scan Y           ; Only in self-controlled mode ; TSC[3:02h].Bit7
          5 = Scan Z1/Z2       ;/                             ;/is sent
```

```
         6 = VBAT measurement
         7 = AUX2 measurement
         8 = AUX1 measurement
         9 = Auto scan. Sequence used is AUX1, AUX2, VBAT.
             Each of these inputs can be enabled or disabled independently
             using TSC[3:13h], and with that sequence is modified accordingly.
             Scan continues until stop bit TSC[3:02h].Bit7 is sent,
             or Bit5-2 of this register are changed.
        10 = TEMP1 measurement
        11 = Port scan: AUX1, AUX2, VBAT
        12 = TEMP2 measurement
        13-15 = Reserved. Do not write these sequences to these bits.
1-0   Interrupt pin (GPIO1 or GPIO2 pin)
         0 = PEN-interrupt    /PENIRQ (active low)
         1 = Data-available   /DATA_AVA (active low)
         2 = PEN-interrupt    PENIRQ and Data-available DATA_AVA (active high)
         3 = Reserved
```

TSC[3:04h] - Precharge and Sense (00h)

```
7     Pen touch detection (0=Enabled, 1=Disabled)
6-4   Precharge time before touch detection
      (0..7 = 0.25us, 1us, 3us, 10us, 30us, 100us, 300us, 1000us)
      (when TSC[3:11h] set to 8MHz)
3     Reserved. Write only zero to this bit.
2-0   Sense time during touch detection
      (0..7 = 1us, 2us, 3us, 10us, 30us, 100us, 300us, 1000us)
      (when TSC[3:11h] set to 8MHz)
```

TSC[3:05h] - Panel Voltage Stabilization (00h)

```
7-6   SAR comparator bias current (0=Normal, 1..3=Increase by 25%, 50%, 100%)
      (use Increase to support higher conversion clock)
5     Sample duration (0=Default, 1=Doubled; for higher impedance)
4-3   Reserved. Write only zeroes to these bits.
2-0   Panel voltage stabilization time before conversion
      (0..7 = 0.25us, 1us, 3us, 10us, 30us, 100us, 300us, 1000us)
      (when TSC[3:11h] set to 8MHz)
```

TSC[3:06h] - Voltage Reference (20h)

```
7     Reference for Non-touch-screen Measurement  (0=External, 1=Internal)
6     Internal reference voltage   (0=1.25V, 1=2.5V)
5     Internal reference powered   (0=Always, 1=Only during conversion)
4     Reserved
3-2   Reference Stabilization Time before Conversion
      (0=0us, 1=100us, 2=500us, 3=1ms) (when TSC[3:11h] set to 8MHz)
1     Reserved
0     Battery measurement input (0=VBAT<=VREF, 1=VBAT=BAT)
```

TSC[3:07h,08h] - Reserved (xxh)

```
7-0   Reserved. Write only the reset value to these bits.
```

TSC[3:09h] - Status Bits 1 (40h) (R)

```
7     Pen Touch detected          (0=Not detected, 1=Detected) (R)
6     ADC Ready                   (0=Busy, 1=Ready)            (R)
5     New data is available       (0=None, 1=Yes)              (R)
4     Reserved. Write only the reset value to this bit.
3     New X data is available     (0=None, 1=Yes)              (R)
2     New Y data is available     (0=None, 1=Yes)              (R)
1     New Z1 data is available    (0=None, 1=Yes)              (R)
0     New Z2 data is available    (0=None, 1=Yes)              (R)
```

Bit0-3 and Bit5 are not valid for the buffer mode.
Bit0-3 are cleared after reading the corresponding data.
Bit5 is cleared after completely reading ALL data.

TSC[3:0Ah] - Status Bits 2 (00h) (R)

```
7     New AUX1 data is available  (0=None, 1=Yes)             (R)
6     New AUX2 data is available  (0=None, 1=Yes)             (R)
5     New VBAT data is available  (0=None, 1=Yes)             (R)
4-2   Reserved. Write only zeros to these bits.
1     New TEMP1 data is available (0=None, 1=Yes)             (R)
0     New TEMP2 data is available (0=None, 1=Yes)             (R)
```

Bit0-1 and Bit5-7 are not valid for the buffer mode.
Bit0-1 and Bit5-7 are cleared after reading the corresponding data.

TSC[3:0Bh,0Ch] - Reserved (xxh)

```
7-0   Reserved. Write only the reset value to these bits.
```

TSC[3:0Dh] - Buffer Mode (03h)

```
7     Buffer Mode Enable (0=Disabled, 1=Enabled)
        (when disabled: RDPTR/WRPTR/TGPTR are set to their default values)
6     Buffer Mode Type   (0=Countinuos-conversion, 1=Single-shot)
5-3   Trigger level for conversion "(N+1)*8*number of converted data"
        0..7 = (8..64)*number of converted data
        uh, does "X*number of converted data" mean "after X conversions"?
2     Reserved
1     Buffer Full  (0=No, 1=Full; contains 64 unread converted data)   (R)
0     Buffer Empty (0=No, 1=Empty; contains 0 unread converted data)   (R)
```

However, in DSi this register is unused, and DSi does instead use TSC[3:0Eh].

TSC[3:0Eh] - Reserved / Undocumented (read by DSi for Pen Down Test) (0Fh)

```
7-0   Reserved. Write only the reset value to these bits.
```

However, in DSi this is used as so... (seems to resemble TSC[3:0Dh])...

```
7     Undoc Enable   (0=Disabled, 1=Enabled)  (R/W)
6     Undoc Whatever (0=Normal)               (R/W)
5-3   Undoc Whatever (5=Normal)               (R/W)
2     Undoc Unused                            (?)
1     Undoc Pendown/DataAvailable?            (R?)
0     Undoc Unused                            (R?)
```

TSC[3:0Fh] - Scan Mode Timer (40h)

```
7     Programmable delay for Touch-screen measurement (0=Disable, 1=Enable)
6-4   Programmable interval timer delay
        (0..7 = 8ms, 1ms, 2ms, 3ms, 4ms, 5ms, 6ms, 7ms)
        (when TSC[3:10h] set to 1MHz)
3     Programmable delay for Non-touch-screen auto measurement   (1=Enable)
2-0   Programmable interval timer delay (0..7 = 1.12min, 3.36min,
        5.59min, 7.83min, 10.01min, 12.30min, 14.54min, 16.78min)
        (uh, what is that? minutes? minimum? or what?)
        (when TSC[3:10h] set to 1MHz)
```

These delays are from the end of one data set of conversion to the start of another new data set of conversion.
Bit7: This interval timer mode is for all self-controlled modes. For host-controlled mode, it is valid only for (X/Y) or (X/Y/Z1/Z2) conversions.

TSC[3:10h] - Scan Mode Timer Clock (81h)

```
7     Clock used for Programmable Delay Timer (0=Internal Osc/8, 1=Ext. MCLK)
6-0   MCLK Divider to Generate 1-MHz Clock for the Programmable Delay Timer
      (1..127=Div1..127, or 0=Div128)
```

The timings marked "(when TSC[3:10h] set to 1MHz)" are assuming the MCLK division result to be 1MHz (1us). Other divider settings will cause those timings to change. Using Internal Osc/8 (Bit7=0) results in 1.025MHz (0.97us), which is almost same as the "1MHz" timings (internal osc isn't too accurate though).
Bit7: External clock is used only to control the delay programmed between the conversions and not used for doing the actual conversion. This is supported to get an accurate delay, because the internal oscillator frequency

varies from device to device.

TSC[3:11h] - SAR ADC Clock (81h)

```
7     Clock used for SAR ADC and TSC FSM (0=Internal Osc/1, 1=External MCLK)
6-0   MCLK Divider for the SAR (min 40ns)  (1..127=Div1..127, or 0=Div128)
```

The timings marked "(when TSC[3:11h] set to 8MHz)" are assuming the MCLK division result to be 8MHz (125ns). Other divider settings will cause those timings to change. For the SAR unit, the division result should be max 25MHz (min 40ns). Using Internal Osc/1 (Bit7=0) results in 8.2MHz (122ns), which is almost same as the "8MHz" timings (internal osc isn't too accurate though).

TSC[3:12h] - Debounce Time for Pen-Up Detection (00h)

```
7     Interface used for the buffer data reading   (0=SPI, 1=I2C)
6     SAR/buffer data update is,
        0: held automatically (to avoid simultaneous buffer read and write
             operations) based on internal detection logic.
        1: held using software control and TSC[3:12h].Bit5.
5     SAR/buffer data update is (only if above Bit6=1),
         0: enabled all the time
         1: stopped so that user can read the last updated data
               without any data corruption.
4-3   Reserved. Write only zeros to these bits.
2-0   Pen-touch removal detection with debounce
        (0..7 = 0us, 8us, 16us, 32us, 64us, 128us, 256us, 512us)
        (when TSC[3:10h] set to 1MHz)
```

The clock used for the debounce has a clock period = debounce duration/8.

TSC[3:13h] - Auto AUX Measurement Selection (00h)

```
7     Auto AUX1 measurement during auto non-touch screen scan (0=Off, 1=On)
6     Auto AUX2 measurement during auto non-touch screen scan (0=Off, 1=On)
5     Auto VBAT measurement during auto non-touch screen scan (0=Off, 1=On)
4     Auto TEMP measurement during auto non-touch screen scan (0=Off, 1=On)
3     TEMP Measurement (0=Use TEMP1, 1=Use TEMP2)
2     AUX1 Usage (0=Voltage measurement, 1=Resistance measurement)
1     AUX2 Usage (0=Voltage measurement, 1=Resistance measurement)
0     Resistance measurement bias (0=Internal bias, 1=External bias)
```

TSC[3:14h] - Touch-Screen Pen Down (00h)

```
7-3   Reserved
2-0   Debounce Time for Pen-Down Detection
        (0..7 = 0us, 64us, 128us, 256us, 512us, 1024us, 2048us, 4096us)
        (when TSC[3:10h] set to 1MHz)
```

The clock used for the debounce has a clock period = debounce duration/8.

TSC[3:15h] - Threshold Check Flags Register (00h) (R)

```
7-6   Reserved. Write only zeros to these bits.
5     AUX1 Maximum (0=Inrange, 1=Exceeds Limit; Equal/Above MAX)
4     AUX1 Minimum (0=Inrange, 1=Exceeds Limit; Equal/Below MIN)
3     AUX2 Maximum (0=Inrange, 1=Exceeds Limit; Equal/Above MAX)
2     AUX2 Minimum (0=Inrange, 1=Exceeds Limit; Equal/Below MIN)
1     TEMP Maximum (0=Inrange, 1=Exceeds Limit; Equal/Above MAX)
0     TEMP Minimum (0=Inrange, 1=Exceeds Limit; Equal/Below MIN)
```

Sticky flag bIts. These are read-only bits. They are automatically cleared once they are read and are set only if the source trigger occurs again.

TSC[3:16h,17h] - AUX1 Maximum Value Check MSB,LSB (0000h)
TSC[3:18h,19h] - AUX1 Minimum Value Check MSB,LSB (0000h)
TSC[3:1Ah,1Bh] - AUX2 Maximum Value Check MSB,LSB (0000h)
TSC[3:1Ch,1Dh] - AUX2 Minimum Value Check MSB,LSB (0000h)
TSC[3:1Eh,1Fh] - Temperature(TEMP1/TEMP2) Maximum Value Check MSB,LSB (0000h)
TSC[3:20h,21h] - Temperature(TEMP1/TEMP2) Minimum Value Check MSB,LSB (0000h)

```
  15-13  Reserved
  12     Threshold check (0=Disabled, 1=Enabled)
         (valid for auto/non-auto scan measurement).
  11-0   Threshold code (12bit)
```

TSC[3:2Ah,2Bh] - Touchscreen X-Coordinate Data MSB,LSB (0000h) (R)
TSC[3:2Ch,2Dh] - Touchscreen Y-Coordinate Data MSB,LSB (0000h) (R)
TSC[3:2Eh,2Fh] - Touchscreen Z1-Pressure Register MSB,LSB (0000h) (R)
TSC[3:30h,31h] - Touchscreen Z2-Pressure Register MSB,LSB (0000h) (R)
TSC[3:36h,37h] - AUX1 Data MSB,LSB (0000h) (R)
TSC[3:38h,39h] - AUX2 Data MSB,LSB (0000h) (R)
TSC[3:3Ah,3Bh] - VBAT Data MSB,LSB (0000h) (R)
TSC[3:42h,43h] - Temperature TEMP1 Data Register MSB,LSB (0000h) (R)
TSC[3:44h,45h] - Temperature TEMP2 Data Register MSB,LSB (0000h) (R)

```
  15-0   Data... but, seems to be always zero on DSi?
```

Going by the datasheet, these registers should contain current sample values, but they seem to be always zero on DSi.
Instead, the samples for the currently selected Conversion Mode can be read from the buffer at TSC[FCh:01h,02h], using the buffer should also ensure that MSB/LSB-pairs won't change within 16bit reads.

TSC[3:22h...29h] - Reserved (xxh)
TSC[3:32h...35h] - Reserved (xxh)
TSC[3:3Ch...41h] - Reserved (xxh)
TSC[3:46h...7Fh] - Reserved (xxh)

```
  7-0    Reserved. Write only the reset value to these bits.
```

TSC[FCh:01h,02h] - Buffer Mode Data MSB,LSB (xxxxh) (R)

```
  15    Ring-buffer Full  (1=All 64 entries are unread)
  14    Ring-buffer Empty (1=All 64 entries are read)
  13    Reserved          (uh?)
  12    Data ID           (0=X/Z1/BAT/AUX2, 1=Y/Z2/AUX1/TEMP)
  11-0  Converted data (12bit), read from "RDPTR" ring-buffer location
```

Note: One can/must read multiple words from index 01h (the index will somewhat automatically toggle between 01h and 02h in that case (or perhaps it does increment to 03h and up, but those indices mirror to 01h and 02h)).

TSC[FCh:03h..7Fh] - Reserved (xxh) (actually mirrors of TSC[FCh:01h,02h])

```
  7-0   Reserved. Write only the reset value to these bits.
```

These are somewhat mirrors of the buffer at TSC[FCh:01h,02h]. Trying to select index=03h starts reading with the 1st buffer byte (ie. it won't start at the 3rd byte).

DSi TSC[04h..05h:xxh], ADC Digital Filter Coefficient RAM

Default values shown for this page only become valid 100us following a hardware or software reset.

TSC[04h-05h:xxh] - ADC Coefficient RAM (126 x 16bit)
Coefficients are signed 16bit (-32,768..+32,767), each occupying 2 bytes (MSB,LSB).
The MSB should always be written first, immediately followed by the LSB (even if only the MSB or LSB portion of the coefficient changes, both registers should be written in this sequence).

```
                  ADC miniDSP   ADC FIR Filter Special
                  Coefficients  Coefficients   Coefficients
  TSC[4:00h]      Page Select   -              -
  TSC[4:01h]      Reserved      -              -
  TSC[4:02h..07h] C1..C3        -              N0,N1,D1 for AGC LPF
                                                 (first-order IIR, used
                                                 as averager to detect level)
  TSC[4:08h..0Dh] C4..C6        -              N0,N1,D1 for ADC-programmable
```

```
                                                 first-order IIR
TSC[4:0Eh..17h]  C7..C11       FIR0..FIR4     N0,N1,N2,D1,D2 for ADC Biquad A
TSC[4:18h..21h]  C12..C16      FIR5..FIR9     N0,N1,N2,D1,D2 for ADC Biquad B
TSC[4:22h..2Bh]  C17..C21      FIR10..FIR14   N0,N1,N2,D1,D2 for ADC Biquad C
TSC[4:2Ch..35h]  C22..C26      FIR15..FIR19   N0,N1,N2,D1,D2 for ADC Biquad D
TSC[4:36h..3Fh]  C27..C31      FIR20..FIR24   N0,N1,N2,D1,D2 for ADC Biquad E
TSC[4:40h..7Fh]  C32..C63      -              -
TSC[5:00h]       Page Select   -              -
TSC[5:01h]       Reserved      -              -
TSC[5:02h..7Fh]  C65..C127     -              -
```

Multi-order FIR Filter (Finite Impulse Response)

```
input[n] ---o--MUL(F0)-->ADD-----------------> output[n]
            |            ^
        input[n-1]       |                          ;\
            |            |                          ; first order FIR
            o--MUL(F1)-->ADD                        ;/
            |            ^
        input[n-2]       |                          ;\
            |            |                          ; second order FIR
            o--MUL(F2)-->ADD                        ;/
            :            ^
            :            :                          ;-further order's
```

First-order IIR Filter (Infinite Impulse Response)

```
input[n] ---o--MUL(N0)-->ADD------------o---> output[n]
            |            ^              |
        input[n-1]       |          output[n-1]     ;\
            |            |              |           ; first order IIR
            o--MUL(N1)-->ADD<--MUL(-D1)--o           ;/
```

Biquad Filter (IIR with 1st and 2nd order)

```
input[n] ---o--MUL(N0)-->ADD------------o---> output[n]
            |            ^              |
        input[n-1]       |          output[n-1]     ;\
            |            |              |           ; first order IIR
            o--MUL(N1)-->ADD<--MUL(-D1)--o           ;/
            |            ^              |
        input[n-2]       |          output[n-2]     ;\
            |            |              |           ; second order IIR
            o--MUL(N2)-->ADD<--MUL(-D2)--o           ;/
```

DSi TSC[08h..0Fh:xxh], DAC Digital Filter Coefficient RAM

Default values shown for this page only become valid 100us following a hardware or software reset.

TSC[08h:01h] - DAC Coefficient RAM Control (00h)

```
7-4   Reserved. Write only the reset value.
3     DAC miniDSP generated flag for toggling MSB of coefficient RAM address
      (only used in non-adaptive mode)                          (R)
2     DAC Adaptive Filtering in DAC miniDSP (0=Disabled, 1=Enabled)   (R/W)
1     DAC Adaptive Filter Buffer Control Flag                   (R)
        aka DAC Coefficient Buffers in adaptive filter mode
           0: miniDSP accesses Buffer A,
                external control interface (=the user?) accesses Buffer B
           1: miniDSP accesses Buffer B,
                external control interface (=the user?) accesses Buffer A
0     DAC Adaptive Filter Buffer Switch Control                 (R/W)
       0: DAC coefficient buffers will not be switched at next frame boundary
```

```
        1: DAC coefficient buffers will     be switched at next frame boundary
           (only if adaptive filtering mode is enabled)
           This bit will self-clear on switching.
```

TSC[08h..0Bh:xxh] - DAC Coefficient RAM, DAC Buffer A (252 x 16bit)
Coefficients are signed 16bit (-32,768..+32,767), each occupying 2 bytes (MSB,LSB).
The MSB should always be written first, immediately followed by the LSB (even if only the MSB or LSB portion of the coefficient changes, both registers should be written in this sequence).

```
                  DAC miniDSP    Special
                 (DAC Buffer A)  DAC-programmable
                  Coefficient    Coefficient
  TSC[8:00h]      Page Select    -
  TSC[8:01h]      Control        - (see above)
  TSC[8:02h..0Bh] C1..C5         N0,N1,N2,D1,D2 for Left Biquad A  ;N0=7FFFh
  TSC[8:0Ch..15h] C6..C10        N0,N1,N2,D1,D2 for Left Biquad B  ;N1,N2,D1,
  TSC[8:16h..1Fh] C11..C15       N0,N1,N2,D1,D2 for Left Biquad C  ;     D2=0
  TSC[8:20h..29h] C16..C20       N0,N1,N2,D1,D2 for Left Biquad D
  TSC[8:2Ah..33h] C21..C25       N0,N1,N2,D1,D2 for Left Biquad E
  TSC[8:34h..3Dh] C26..C30       N0,N1,N2,D1,D2 for Left Biquad F
  TSC[8:3Eh..3Fh] C31            -
  TSC[8:40h..41h] C32            for 3D PGA for PRB_P23, PRB_P24 and PRB_P25
  TSC[8:42h..4Bh] C33..C37       N0,N1,N2,D1,D2 for Right Biquad A
  TSC[8:4Ch..55h] C38..C42       N0,N1,N2,D1,D2 for Right Biquad B
  TSC[8:56h..5Fh] C43..C47       N0,N1,N2,D1,D2 for Right Biquad C
  TSC[8:60h..69h] C48..C52       N0,N1,N2,D1,D2 for Right Biquad D
  TSC[8:6Ah..73h] C53..C57       N0,N1,N2,D1,D2 for Right Biquad E
  TSC[8:74h..7Dh] C58..C62       N0,N1,N2,D1,D2 for Right Biquad F
  TSC[8:7Eh..7Fh] C63            -
  TSC[9:00h]      Page Select    -
  TSC[9:01h]      Reserved       - (do not write to this register)
  TSC[9:02h..07h] C65..C67       N0,N1,D1 for Left first-order IIR
  TSC[9:08h..0Dh] C68..C70       N0,N1,D1 for Right first-order IIR
  TSC[9:0Eh..13h] C71..C73       N0,N1,D1 for DRC first-order high-pass filter
  TSC[9:14h..19h] C74..C76       N0,N1,D1 for DRC first-order low-pass filter
  TSC[9:1Ah..7Fh] C77..C127      -
  TSC[A:00h]      Page Select    -
  TSC[A:01h]      Reserved       - (do not write to this register)
  TSC[A:02h..7Fh] C129..C191     -
  TSC[B:00h]      Page Select    -
  TSC[B:01h]      Reserved       - (do not write to this register)
  TSC[B:02h..7Fh] C193..C255     -
```

TSC[0Ch..0Fh:xxh] - DAC Coefficient RAM, DAC Buffer B (252 x 16bit)
This is essentially same as above Buffer A. But it's unclear if Buffer B is having the same special Biquad/3DPGA/IIR/DRC functions (the official datasheet doesn't mention them, but it does specify the initial reset values same as for Buffer A, ie. with value 7FFFh for the locations that correspond to "N0" coefficients, which is suggesting that those special functions are present in Buffer B, too).

```
                  DAC miniDSP    Special
                 (DAC Buffer A)  DAC-programmable
                  Coefficient    Coefficient
  TSC[C:02h..0Bh] C1..C5         Unknown  ;\
  TSC[C:0Ch..15h] C6..C10        Unknown  ;
  TSC[C:16h..1Fh] C11..C15       Unknown  ; maybe Left Biquad A..F
  TSC[C:20h..29h] C16..C20       Unknown  ; as for Buffer A
  TSC[C:2Ah..33h] C21..C25       Unknown  ;
  TSC[C:34h..3Dh] C26..C30       Unknown  ;/
  TSC[C:3Eh..3Fh] C31            -
  TSC[C:40h..41h] C32            Unknown maybe 3D PGA as for Buffer A
  TSC[C:42h..4Bh] C33..C37       Unknown  ;\
  TSC[C:4Ch..55h] C38..C42       Unknown  ;
  TSC[C:56h..5Fh] C43..C47       Unknown  ; maybe Right Biquad A..F
  TSC[C:60h..69h] C48..C52       Unknown  ; as for Buffer A
  TSC[C:6Ah..73h] C53..C57       Unknown  ;
```

```
TSC[C:74h..7Dh]  C58..C62       Unknown  ;/
TSC[C:7Eh..7Fh]  C63            -
TSC[D:00h]       Page Select    -
TSC[D:01h]       Reserved       - (do not write to this register)
TSC[D:02h..07h]  C65..C67       Unknown  ;\
TSC[D:08h..0Dh]  C68..C70       Unknown  ; maybe IIR and DRC
TSC[D:0Eh..13h]  C71..C73       Unknown  ; as for Buffer A
TSC[D:14h..19h]  C74..C76       Unknown  ;/
TSC[D:1Ah..7Fh]  C77..C127      -
TSC[E:00h]       Page Select    -
TSC[E:01h]       Reserved       - (do not write to this register)
TSC[E:02h..7Fh]  C129..C191     -
TSC[F:00h]       Page Select    -
TSC[F:01h]       Reserved       - (do not write to this register)
TSC[F:02h..7Fh]  C193..C255     -
```

DSi TSC[20h..2Bh:xxh], TSC[40h..5Fh:xxh] ADC/DAC Instruction RAM

TSC[20h..2Bh:xxh] - ADC DSP Engine Instruction RAM (384 x 24bit)
ADC miniDSP Instructions are 20bit, each occupying 3 bytes (MSB,MID,LSB) (with dummy padding in upper 4bit of MSB).

```
TSC[20h..2Bh:00h]       Page Select
TSC[20h..2Bh:01h]       Reserved
TSC[20h:02h...61h]      ADC Instructions 0...31
TSC[21h:02h...61h]      ADC Instructions 32...63
TSC[22h:02h...61h]      ADC Instructions 64...95
TSC[23h:02h...61h]      ADC Instructions 96...127
TSC[24h:02h...61h]      ADC Instructions 128...159
TSC[25h:02h...61h]      ADC Instructions 160...191
TSC[26h:02h...61h]      ADC Instructions 192...223
TSC[27h:02h...61h]      ADC Instructions 224...255
TSC[28h:02h...61h]      ADC Instructions 256...287
TSC[29h:02h...61h]      ADC Instructions 288...319
TSC[2Ah:02h...61h]      ADC Instructions 320...351
TSC[2Bh:02h...61h]      ADC Instructions 352...383
TSC[20h..2Bh:62h..7Fh]  Reserved
```

TSC[40h..5Fh:xxh] - DAC DSP Engine Instruction RAM (1024 x 24bit)
DAC miniDSP Instructions are 24bit (uh, unlike 20bit ADC ones?), each occupying 3 bytes (MSB,MID,LSB).

```
TSC[40h..5Fh:00h]       Page Select
TSC[40h..5Fh:01h]       Reserved
TSC[40h:02h...61h]      DAC Instructions 0...31
TSC[41h:02h...61h]      DAC Instructions 32...63
TSC[42h:02h...61h]      DAC Instructions 64...95
TSC[43h:02h...61h]      DAC Instructions 96...127
TSC[44h:02h...61h]      DAC Instructions 128...159
TSC[45h:02h...61h]      DAC Instructions 160...191
TSC[46h:02h...61h]      DAC Instructions 192...223
TSC[47h:02h...61h]      DAC Instructions 224...255
TSC[48h:02h...61h]      DAC Instructions 256...287
TSC[49h:02h...61h]      DAC Instructions 288...319
TSC[4Ah:02h...61h]      DAC Instructions 320...351
TSC[4Bh:02h...61h]      DAC Instructions 352...383
TSC[4Ch:02h...61h]      DAC Instructions 384...415
TSC[4Dh:02h...61h]      DAC Instructions 416...447
TSC[4Eh:02h...61h]      DAC Instructions 448...479
TSC[4Fh:02h...61h]      DAC Instructions 480...511
TSC[50h:02h...61h]      DAC Instructions 512...543
TSC[51h:02h...61h]      DAC Instructions 544...575
```

```
TSC[52h:02h...61h]      DAC Instructions 576...607
TSC[53h:02h...61h]      DAC Instructions 608...639
TSC[54h:02h...61h]      DAC Instructions 640...671
TSC[55h:02h...61h]      DAC Instructions 672...703
TSC[56h:02h...61h]      DAC Instructions 704...735
TSC[57h:02h...61h]      DAC Instructions 736...767
TSC[58h:02h...61h]      DAC Instructions 768...799
TSC[59h:02h...61h]      DAC Instructions 800...831
TSC[5Ah:02h...61h]      DAC Instructions 832...863
TSC[5Bh:02h...61h]      DAC Instructions 864...895
TSC[5Ch:02h...61h]      DAC Instructions 896...927
TSC[5Dh:02h...61h]      DAC Instructions 928...959
TSC[5Eh:02h...61h]      DAC Instructions 960...991
TSC[5Fh:02h...61h]      DAC Instructions 992...1023
TSC[40h..5Fh:62h..7Fh]  Reserved
```

The miniDSP instruction set isn't officially documented anywhere. Texas Instruments has merely released an "assembler" for the miniDSP (that is, a graphical drag-and-drop utility referred to as PurePath Studio).

DSi I2C Bus

I2C Bus
DSi I2C I/O Ports
DSi I2C Signals

Device 4Ah (BPTWL chip) (LED/Volume/Powerbutton/Reset)
DSi I2C Device 4Ah (BPTWL chip)

Device 78h/7Ah (Aptina MT9V113 Cameras)
DSi Aptina Camera Initialization
Directly addressed I2C Registers (16bit index, 16bit data):
DSi Aptina Camera Registers: SYSCTL (0000h-0051h)
DSi Aptina Camera Registers: RX_SS, FUSE, XDMA (0100h-099Fh)
DSi Aptina Camera Registers: CORE (3000h-31FFh, 38xxh)
DSi Aptina Camera Registers: SOC1 (3210h-33FDh)
DSi Aptina Camera Registers: SOC2 (3400h-3729h)
Indirectly addressed MCU Variables (via above "XDMA" commands):
DSi Aptina Camera Variables: RAM/SFR/MON (GPIO/Monitor) (MCU:0000h-20xxh)
DSi Aptina Camera Variables: SEQ (Sequencer) (MCU:21xxh)
DSi Aptina Camera Variables: AE (Auto Exposure) (MCU:22xxh)
DSi Aptina Camera Variables: AWB (Auto White Balance) (MCU:23xxh)
DSi Aptina Camera Variables: FD (Anti-Flicker) (MCU:24xxh)
DSi Aptina Camera Variables: MODE (Mode/Context) (MCU:27xxh)
DSi Aptina Camera Variables: HG (Histogram) (MCU:2Bxxh)
I2C Bus Caution: The Camera I2C access requires the "16.76MHz Camera External Clock" enabled in Port 4004004h.Bit8 on ARM9 Side. For accessing registers other than SYSCTL/CORE, one must also clear the Standby flag in SYSCTL[0018h].

Device A0h/E0h (Unknown, maybe cameras from other manufacturer)
DSi Alternate Cameras from Unknown Manufacturer

Camera Data Transfers
Camera configuration is done on ARM7 side via serial I2C bus. However, the actual Camera Data transfers are done on ARM9 side through 8bit parallel bus:
DSi Cameras

Broken Cameras (defunct device 78h/7Ah/A0h/E0h)
Consoles should contain two Aptina cameras, or two Unknown cameras. The Unlaunch installer is throwing a warning when detecting Unknown cameras, so far nobody has reported that case, so Unknown cameras seem to be very rare (if they were ever used at all). However, two people reported that warning showing FFh's in the ID bytes for most or all cameras (caused by a broken camera with only one working camera, or broken camera connector with no working cameras at all).
Bottom line is that broken cameras are more common than unknown cameras, and games with optional/extra camera features should support that situation: ie. disable the feature in case of broken camera(s) instead of becoming unplayable.

Device 90h (Whatever)
Trying to read IC2 for this device just returns FFh? Maybe exists in debug version only. The firmware contains a few functions for accessing this register.

```
Register  Width   Description
02h       1       Used for DSi IRQ6 IF flags
                    uh, IF.Bit6 would be Timer3overflow ?
                    or, IF2.Bit6 would be PowerButton ?
04h       1       Unknown (bit0 toggled)
```

Unknown purpose.

Device 40h (Whatever) (and New3DS QTM)
Trying to read IC2 for this device just returns FFh? Maybe exists in debug version only. The firmware doesn't use this register (it does only contain the device number in the device table).
Caution: New3DS does use device number 40h for the QTM IO Expander, this device will be visible in DSi mode (and shouldn't be mistaken for whatever DSi device 40h was originally used for).

DSi I2C Devices

```
Device  Delay      Description
7Ah     0       0  Camera0(internal)  ;Aptina MT9V113 (SelfPortrait)
78h     0       1  Camera1(external)  ;Aptina MT9V113 (External)
A0h     0       2  Camera0 config (Ext) ;\maybe for other manufacturer?
E0h     0       3  Camera1 config (Self);/
4Ah     180h    4  BPTWL Chip (LED/Volume/Powerbutton/Reset)
40h     0       5  Debug?
90h     0       6  Debug?
```

Delay: required swiWaitByLoop delay

DSi Secondary I2C Devices
There are also some internal/secondary I2C busses (not connected to the ARM CPUs).

```
xxh   Power Managment Device                        (connected to BPTWL chip)
50h   I2C bus potentiometer (volume D/A converter) (connected to BPTWL chip)
A0h   I2C bus EEPROM                          (connected to Atheros wifi chip)
-     I2C voltage translator                  (between ARM CPU and BPTWL chip)
```

DSi I2C I/O Ports

4004500h - DSi7 - I2C_DATA (R/W)

```
0-7  Data (or Device, or Register)
```

When sending data, I2C_DATA should be written <before> setting I2C_CNT.bit7.
When reading data, I2C_DATA should be read <after> I2C_CNT.bit7 goes off.
Alongsides with the 8bit data, an additional 1bit "Ack" flag is transferred as response to the data (ie. in opposite direction of data direction), the Ack is located in I2C_CNT.Bit4, and it's usually indicating errors (or in some cases it appears to be also used to indicate that no further data is to be transferred).

4004501h - DSi7 - I2C_CNT (R/W)

```
0    Stop  (0=No, 1=Stop/last byte)
1    Start (0=No, 1=Start/first byte)
2    Error (0=No, 1=Pause/Flush? after Error, used with/after Stop)
3    Unknown/unused (0)
4    Ack Flag         (0=Error, 1=Okay)  (For DataRead: W, for DataWrite: R)
5    Data Direction   (0=Write, 1=Read)
6    Interrupt Enable (0=Disable, 1=Enable)
7    Start/busy       (0=Ready, 1=Start/busy)
```

I2C Transfer Flowchart

The first byte (with the "Start" condition) contains the device number, which consists of a 7bit chip ID and a direction flag in bit0 (0=Write or 1=Read). The direction flag applies to all following bytes (until last byte with "Stop" condition), that rule means that "Write Index & Write Data" can be done in a single step, whilst "Write Index & Read Data" must be split into two separate steps (each with own "Start/Stop" conditions):

```
For Writing:
 Write Device+0 (with Start condition)                ;\
 Write Index byte(s)                                  ; write index + data
 Write Data byte(s) (last byte with Stop condition)    ;/
For Reading:
 Write Device+0 (with Start condition)                ;\1st step: write index
 Write Index byte(s) (last byte with Stop condition)  ;/
 Write Device+1 (with Start condition)                ;\2nd step: read data
 Read Data byte(s) (last byte with Stop condition)     ;/
```

The index is usually a single byte (except for Aptina cameras, which do use 16bit indices transferred in two bytes; ordered MSB, LSB).

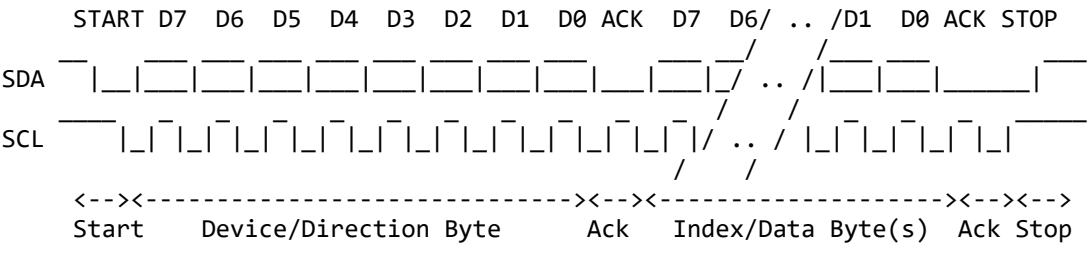
Per-byte transfer completion is indicated by the Start/busy flag, which should also indicate if the I2C chip is ready for next byte (I2C devices can hold the clock line low if they aren't ready), however, the BPTWL chip somehow doesn't support that, and it should be accessed with an extra delay:

```
Invoke byte-transfer
Do WaitByLoop (needed for the BPTWL device only)
Wait for start/busy flag to get zero
```

Note: The DSi firmware is doing eight retries per I2C command (in case of receiving wrong ACK bytes), unknown if that's really required, a stable system should never need to do retries.

DSi I2C Signals

Below is some pseudo code for the I2C signal transmission. The DSi hardware is doing most of that stuff automatically. The pseudo code may be useful for understanding the purpose of the start/stop/ack flags in the control register.



SDA should be changed at/after falling SCL edge (except for Start/Stop conditions, which are output during SCL=High).

The Device/Direction Byte is sent by master (the byte contains a 7bit device address in bit7-1, and a direction flag in bit0). The direction of the follwing index/data byte(s) depends on that direction flag (0=Write, 1=Read). The ACK bit responses are sent in opposite direction as the preceeding byte.

The SCL line is driven by master, however, when the master changes SCL from Low to HighZ, then the slave

may keep SCL held Low to signalize that it isn't yet ready for the next bit.

i2c_write_byte(send_start,send_stop,databyte):

```
  if (send_start) then i2c_start_cond()                    ;-start (if so)
  for i=7 downto 0, i2c_write_bit(databyte.bit(i)), next i  ;-write 8bit
  nack = i2c_read_bit()                                    ;-read nack
  if (send_stop) then i2c_stop_cond()                      ;-stop (if so)
  return nack        ;return 0 if ack by the slave.        ;-return nack
```

i2c_read_byte(nack,send_stop):

```
  for i=7 downto 0, databyte.bit(i)=i2c_read_bit(), next i  ;-read 8bit
  i2c_write_bit(nack)                                      ;-write nack
  if (send_stop) then i2c_stop_cond()                      ;-stop (if so)
  return databyte                                          ;-return databyte
```

i2c_write_bit(bit):

```
  if (bit) then SDA=HighZ else SDA=Low          ;-
  I2C_delay()                                   ;-
  SCL=HighZ
  wait until SCL=High (or timeout)              ;-wait (for clock stretching)
  if (bit=1 and SDA=Low) then arbitration_lost();-errif other HW pulls SDA=low
  I2C_delay()                                   ;-
  SCL=Low                                       ;-
```

i2c_read_bit():

```
  SDA=HighZ                                     ;-let the slave drive data
  I2C_delay()                                   ;-delay (one half clk)
  SCL=HighZ
  wait until SCL=High (or timeout)              ;-wait (for clock stretching)
  bit = SDA                     ;-
  I2C_delay()                   ;-delay (one half clk)
  SCL=Low                       ;-
  return bit                    ;-
```

i2c_start_cond():

```
  if (started) then            ;if started, do a restart cond
    SDA=HighZ                   ;-set SDA to 1
    I2C_delay()
    SCL=HighZ
    wait until SCL=High (or timeout)         ;-wait (for clock stretching)
    I2C_delay()         ;Repeated start setup time, minimum 4.7us
  if (SDA=Low) then arbitration_lost()
  SDA=Low                       ;-
  I2C_delay()
  SCL=Low
  started = true
```

i2c_stop_cond():

```
  SDA=Low                               ;-
  I2C_delay()                           ;-
  SCL=HighZ                             ;-
  wait until SCL=High (or timeout)      ;-wait (for clock stretching)
  I2C_delay()   ;Stop bit setup time, minimum 4us
  if (SDA=Low) then arbitration_lost()
  I2C_delay()
  started = false
```

DSi I2C Device 4Ah (BPTWL chip)

I2C Bus Caution: The BPTWL chip requires swiWaitByLoop(180h) after each I2C byte transfer (if the

Version/Speed byte at BPTWL[00h] indicates "Fast", then the delay can be reduced to 90h instead of 180h). And, SPI bus Powerman chip does somehow interact with I2C BPTWL chip; there must be a similar delay between Powerman writes and BPTWL writes.

BPTWL Chip (LED/Volume/Powerbutton/Reset) (Device 4Ah)

```
  00h     R   Version/Speed (usually 33h) (00h..20h=Slow, 21h..FFh=Fast)
  01h     R   Unknown (00h)
  02h     R   Unknown (50h)
  03h-0Fh -   Reserved (5Ah-filled)
  10h     R   Power Button Status (bit0=WasWhat?, bit1=IsDown, bit3=WasDown?)
                (bit0/3 are cleared after reading)
  11h     R/W Reset (00h=No, 01h=Force Reset, 02h=???)
  12h     R/W Power Button Tapping (00h=Auto-Reset, 01h=IRQ)
  13h-1Fh -   Reserved (5Ah-filled)
  20h     R   Battery State (bit0..3=Battery Level, bit7=Charge)
          W   On 3DS in DSi mode: Write 8=Shutdown, 4=Return to 3DS mode ?
  21h     R/W Unknown (07h)
  22h-2Fh -   Reserved (5Ah-filled)
  30h     R/W Wifi LED (0/2=Off, 1=On, 3=BlinkOnTraffic) and bit4=SDIO enable
  31h     R/W Camera LED (00h=Off, 01h=On, 02h=Blink)
  32h-3Fh -   Reserved (5Ah-filled)
  40h     R/W Volume Level (00h..1Fh)             ;\nonvolatile!
  41h     R/W Backlight Level (00h..04h)          ;/
  42h-5Fh -   Reserved (5Ah-filled)
  60h     ??  Unknown (00h) FFh: Disable I2C reading, and Purple Power LED?
  61h     R   Unknown (01h)
  62h     R   Unknown (50h)
  63h     R/W Unknown (00h) FFh: Purple Power LED (red+blue on)
  64h-6Fh -   Reserved (5Ah-filled)
  70h     R/W Bootflag (00h=Coldboot, 01h=Warmboot/SkipHealthSafety)
  71h     R/W Unknown (00h)
  72h-77h R/W Unknown (00h-filled)
  78h-7Fh -   Reserved (5Ah-filled)
  80h     R/W Unknown (10h)     ;\can affect/disable Power Button Tapping
  81h     R/W Unknown (64h)     ;/
  82h-FFh -   Reserved (5Ah-filled)
```

The R/W values can be set to 00h..FFh (except Index 40h/41h are quickly cropped to max 1Fh/04h, and Index 70h/71h are (after some time) cropped to 01h/02h).
Above should probably also include:

```
  Forced volume (for alerts) (ie. alternately to current "user volume")
```

BPTWL Emulation on 3DS

The 3DS MCU emulates the BPTWL chip for DSi mode (and the emulated registers can be also accessed in 3DS mode). The emulation supports only a few of the original BPTWL registers & bits though:

```
  00h     R   Version/Speed (35h on New3DS)
  10h     R   Power Button Status (only 2 bits?: bit3=WasDown?, bit6=?)
  11h     R/W Reset (01h=Reset, other=ignored) (read: always 00h)
  12h     R/W Power Button Tapping (bit0,1,7=?)
  20h     R   Battery State
  31h     R/W Camera LED (bit0,1)
  40h     R/W Volume Level
  70h     R/W Bootflag
  Other   -   Unused (FFh)
```

DSi Power Button

Logically, the Power Button has two functions:

```
  Short tap --> reset (warmboot, go to DSi menu, without health and safety)
  Hold 1 second --> power-off
```

Technically, the button can have three functions:

```
  Auto-Reset (used for NDS games)
  IRQ (supposed to be used with Manual-Reset) (used for DSi games)
  Forced Power-off (for games which fail to handle the IRQ within 5 seconds)
```

DSi games should handle the IRQ as follows: First, do some clean up (like finishing writes to SD/MMC storage; to avoid FAT corruption). Then, issue a Reset manually (via I2C/BPTWL registers [70h]=01h/Warmboot, [11h]=01h/Reset).
Power-Off can be implemented via SPI/Power Managment Device, however, games only need to implement Manual-Reset or Auto-Reset, but don't need to implement Power-Off (the firmware will do that automatically if the button is held down for 1 second after issuing the Reset).
Ideally, emulators should also reproduce the power button (when resetting or closing the emulator): Signalize power-button and keep the emulation running until the game responds by Reset (or until five second timeout). That will allow the game to finish writes to emulated SD/MMC storage.

DSi Autostart on Warmboot (20h-byte area) (also requires BPTWL[70h]=01h)
The DSi can be commanded to load a different title (eg. System Settings), instead of showing the Boot Menu after warmboot.
DSi Autoload on Warmboot

Older blurb...
```
0x10    1       Power flags. When bit0 is set, arm7 does a system reset.
                  When bit1 or bit3 are set, arm7 does a shutdown. Bits 0-2
                  are used for DSi IRQ6 IF flags (uh, rather IF2 maybe?).
0x20    1       Battery flags. When zero the battery is at critical level,
                  arm7 does a shutdown. Bit7 is set when the battery is
                  charging. Battery levels in the low 4-bits: battery icon
                  bars full 0xF, 3 bars 0xB, 2 bars 0x7, one solid red bar
                  0x3, and one blinking red bar 0x1. When plugging in or
                  removing recharge cord, this value increases/decreases
                  between the real battery level and 0xF, thus the battery
                  level while bit7 is set is useless.
```

BPTWL/BPUTL Chip Names
```
DSi:      Renesas Electronics "BPTWL, KG07K"           ;reg[00h]=33h
DSiXL:    Renesas Electronics "BP UTL-1, KG08"         ;reg[00h]=BBh or B7h
3DS:      Renesas Electronics "UC CTR, 041KM73, KG10"  ;reg[00h]=?
New3DSXL: Renesas Electronics "UC KTR, 423KM01, 'TK14" ;reg[00h]=35h
```

DSi Autoload on Warmboot

Overview
Launcher (and unlaunch) can be commanded to autoload a different title.
```
2000000h  Autoload Parameters for newly loaded title  ;<-- optional extra
2000300h  Autoload via numeric Title ID               ;<-- official method
2000800h  Autoload via string "device:\path\filename" ;<-- alternate method
2FFD800h  Title List (jump-able titles for use at 2000300h)
BPTWL[70h].bit0  Warmboot flag                        ;<-- required flag
BPTWL[11h].bit0  Trigger reset                        ;<-- trigger reset
```
Moreover, autoload can occur on warmboot & coldboot: Launcher autoloads any ROM cartridge with carthdr[01Fh].Bit2. Unlaunch defaults to autoload any file named "sdmc:\bootcode.dsi".

Examples
DSi Browser settings screen allows to autoload System Settings (via 2000300h), and automatically enter the Internet options page (via 2000000h).
Nintendo Zone, if started with Wireless Communications disabled, allows to reboot itself (via specifying it's own Title ID in 2000300h).
Homebrew frontends for unlaunch could start themselves (via bootcode.dsi) and then command unlaunch to load another title (via 2000800h or 2000300h).

2000000h - Optional Auto-load parameters (passed on to new title)

```
  2000000h 8    AutoParam Old Title ID (former title)    ;carthdr[230h]
  2000008h 1    AutoParam Unknown/Unused
  2000009h 1    AutoParam Flags (03h=Stuff is used?)
  200000Ah 2    AutoParam Old Maker code                 ;carthdr[010h]
  200000Ch 2    AutoParam Unknown (02ECh) ;\counter/length/indices/whatever?
  200000Eh 2    AutoParam Unknown (0000h) ;/
  2000010h 2    AutoParam CRC16 on [000h..2FFh], initial=FFFFh, [010h]=0000h
  2000012h 2    AutoParam Unknown/Unused (000Fh = want Internet Settings?)
  2000014h 2ECh AutoParam Unknown... some buffer... string maybe?
```
Above is the overall skeleton as intended by Nintendo, the purpose/format of the 2ECh bytes is unknown (there seems to be some relation to entries [0Ch] and [0Eh], but theoretically, each title could use that bytes as pleased).

2000300h - Nintendo Auto-load feature (via numeric Title ID)
```
  2000300h 4    AutoLoad ID ("TLNC")
  2000304h 1    AutoLoad Unknown/unused (usually 01h)
  2000305h 1    AutoLoad Length of data at 2000308h (01h..18h,for CRC,18h=norm)
  2000306h 2    AutoLoad CRC16 of data at 2000308h (with initial value FFFFh)
  2000308h 8    AutoLoad Old Title ID (former title) (can be 0=anonymous)
  2000310h 8    AutoLoad New Title ID (new title to be started,0=none/launcher)
  2000318h 4    AutoLoad Flags (bit0, 1-3, 4, 5,6,7) ;usually 16bit, once 32bit
  200031Ch 4    AutoLoad Unused (but part of checksummed area when CRC len=18h)
  2000320h E0h  AutoLoad Unused (but zerofilled upon erasing autoload area)
```
Flags (usually 13h or 17h):
```
  0     IsValid (somehow enables/disables HealthSafety when TitleID is wrong?)
  1-3   Boottype (01h=Cartridge, 02h=Landing, 03h=DSiware) (see below)
  4     Unknown
  5     Unknown
  6     LoadCompl (causes some error when set) (loading completed flag?)
  7     Unknown
  8-15  Unused
  16-31 Unused (usually not accessed at all, with normal 16bit reads)
```
Boottypes (in Flags.bit1-3):
```
  01h = Cartridge (with NewTitleID)    (with RSA signed header, or Whitelisted)
  02h = Landing ("nand:/tmp/jump.app") (with RSA signed DownloadPlay footer)
  03h = DSiware (with NewTitleID)      (with RSA signed header)
```
TitleID.LSW should match DSi cart header (or be reverse of NDS gamecode?)
TitleID.MSW should match DSi cart header (or be zero for NDS titles?)
Note: Many titles do create the above structure even when not requesting to boot a new file: with NewTitleID=0=none & flags=13h=cartridge (in that case flags should be ignored, and NewTitleID=0=none has priority).

2000800h - Unlaunch Auto-load feature (via "device:/Path/Filename.ext")
```
  2000800h 12   Unlaunch Auto-load ID ("AutoLoadInfo")
  200080Ch 2    Unlaunch Length for CRC16 (fixed, must be 3F0h)
  200080Eh 2    Unlaunch CRC16 (across 2000810h..2000BFFh, initial value FFFFh)
  2000810h 4    Unlaunch Flags
  2000814h 2    Unlaunch Upper screen BG color (0..7FFFh)
  2000816h 2    Unlaunch Lower screen BG color (0..7FFFh)
  2000818h 20h  Unlaunch Reserved (zero)
  2000838h 208h Unlaunch Device:/Path/Filename.ext (16bit Unicode,end by 0000h)
  2000A40h 1C0h Unlaunch Reserved (zero)
```
Unlaunch Flags (usually 01h):
```
  0    Load the title at 2000838h
  1    Use colors 2000814h (use if loaded title is KNOWN to use such colors)
  2-31 Reserved (zero)
```
The name can use slashes or backslashes for folders, and it can use both long or short filenames (LFN or 8.3). The total length should not exceed 260 characters including EOL (alike windows MAX_PATH, although WinNT seems to allow up to 32K characters, which isn't supported here).
```
  "nand:/path/name.ext",0000h   File on 1st partition of internal eMMC
  "sdmc:/path/name.ext",0000h   File on 1st partition of external SD/MMC
```

```
"cart:",0000h      ROM cartridge (in NDS cartridge slot)
"menu:",0000h      Force starting unlaunch filemenu
"sett:",0000h      Force starting unlaunch options menu
"wifi:",0000h      Force starting unlaunch wifiboot overlay
```
Case-sensitivity: device, path and file can use upper/lower case A-Z (not case-sensititive), however currently any other letters are case-sensitive (eg. umlaut's or accented letters) (that is, they must be uppercase for short names, or have matching case for long names).
For black colors, you should also disable backlights before issuing reset (else screen will flash white for a short moment during initial forced blank; before unlaunch gets started).

2FFD800h - Title List
Before autoloading a title, one should make sure that the title is actually installed (and which region code it is having, ie. one should use wildcards that ignore the lower 8bit of the Title ID when searching the title).
The offical way is to look up the Title List in RAM, this is faster than manually crawling the directory tree.
However, there are some restrictions: The Title List contains only titles from the same Maker (as the currently loaded title), plus some special "jumpable" system titles.
```
2FFD800h 1     Titles: Number of titles in below lists (max 76h)
2FFD801h 0Fh   Titles: Zerofilled
2FFD810h 10h   Titles: Pub Flags (1bit each) ;same maker plus public.sav
2FFD820h 10h   Titles: Prv Flags (1bit each) ;same maker plus private.sav
2FFD830h 10h   Titles: Jmp Flags (1bit each) ;jumpable or current-title
2FFD840h 10h   Titles: Mkr Flags (1bit each) ;same maker
2FFD850h 3B0h  Titles: Title IDs (8 bytes each)
```
Related carthdr entries are:
```
[010h].bit0-15  Maker (must match current title for Mkr Flags)
[01Dh].bir0     Jump (must be set for Jmp Flags)
[230h].bit0-63  Title ID (must be nonzero for being listed)
[238h].bit0-31  Public.sav size (must be nonzero for Pub Flags)
[23Ch].bit0-31  Private.sav size (must be nonzero for Prv Flags)
```
The list can contain the hidden Nintendo Zone utility, and DSi ROM cartridges (both provided that Maker does match up with current title).
The jumpable titles with [01Dh].bit0 that are always in the list are:
```
00030015484E42xxh  ;System Settings
00030005484E4441h  ;DS Download Play
00030005484E4541h  ;Pictochat
00030004484E47xxh  ;Nintendo DSi Browser (if installed)
```
The list does NOT contain the Launcher itself, nor files from System Data folder (WifiFirmware, Whitelist, VersionData), nor NDS ROM cartridges, nor anything where Jmp+Mkr flags would be both zero.
If started via 2000300h, then the New Title (from 2000310h) does also receive the the Old Title ID (from 2000308h) with Jmp flag being set; ie. permitting to return to the Old Title (to know which title was the old title, one should probably look at 2000000h, or at 2000308h if that's still intact in memory?).
Also, the Jmp flag is always set for the current title; ie. permitting the title to reboot itself.

DSi Aptina Camera Initialization

Aptina I2C registers are accessed via 16bit index, and one or more data bytes at auto-increasing indices (usually, all transfers are done as big-endian 2-byte (16bit) values at even indices). Additional MCU Variables are accessed indirectly via XMDA registers.

aptina_get_chip_id:
Reading a 16bit value from index 0000h returns the CHIP_VERSION_REG (always 2280h=MT9V113 for DSi/3DS/New3DS), the DSi games are actually reading that register, but they seem to ignore it's value.
If the DSi isn't fitted with Aptina cameras, then reading anything from device 78h/7Ah would most likely return FFh-bytes.

brightness / low light environments

Below configurations are okay for daylight (without much sunshine), but the picture will be almost completely black at night (in rooms with small bulbs). There are probably numerous good/bad ways to manipulate the brightness. Some random solutions are:
Leave AE_MIN_INDEX/AE_MAX_INDEX at their power-on defaults (instead of using below settings) (the power-on defaults will greatly improve the brightness, but the conversion will be also much slower). Increasing COARSE_INTEGRATION_TIME to some big value (like 0800h) does also seem to raise the brightness.

aptina_code_list_init:
Below is some minimal initialization (though it might still include some unnecessary stuff). Most important sections are leaving standby mode, matching PLL to DSi timings, selecting desired resolution(s) and YUV color format.
DSi games are usually initializing further stuff like P0..P4 Coefficients, Gamma Tables, and Color Correction Matrices - but the cameras are also working when leaving those settings at their power-on defaults.
It's recommended to initialize both cameras in parallel (eg. issue Wakeup to <both> cameras, and <then> wait for Wakeup completion; this is faster than doing it separately for each camera).

```
  AptWr     ,0001Ah,00003h   ;RESET_AND_MISC_CONTROL (issue reset)   ;\reset
  AptWr     ,0001Ah,00000h   ;RESET_AND_MISC_CONTROL (release reset) ;/
  AptWr     ,00018h,04028h   ;STANDBY_CONTROL (wakeup)               ;\
  AptWr     ,0001Eh,00201h   ;PAD_SLEW                               ; wakeup
  AptWr     ,00016h,042DFh   ;CLOCKS_CONTROL                         ;
  AptWaitClr,00018h,04000h   ;STANDBY_CONTROL (wait for WakeupDone)  ;
  AptWaitSet,0301Ah,00004h   ;UNDOC_CORE_301A (wait for WakeupDone)  ;/
  AptWrMcu  ,002F0h,00000h   ;UNDOC! RAM?
  AptWrMcu  ,002F2h,00210h   ;UNDOC! RAM?
  AptWrMcu  ,002F4h,0001Ah   ;UNDOC! RAM?
  AptWrMcu  ,02145h,002F4h   ;UNDOC! SEQ?
  AptWrMcu  ,0A134h,  001h   ;UNDOC! SEQ?
  AptSetMcu ,0A115h,  002h   ;SEQ_CAP_MODE (set bit1=video)
  AptWrMcu  ,02755h,00002h   ;MODE_OUTPUT_FORMAT_A (bit5=0=YUV)      ;\select
  AptWrMcu  ,02757h,00002h   ;MODE_OUTPUT_FORMAT_B                   ;/YUV mode
  AptWr     ,00014h,02145h   ;PLL_CONTROL                            ;\
  AptWr     ,00010h,00111h   ;PLL_DIVIDERS                           ; match
  AptWr     ,00012h,00000h   ;PLL_P_DIVIDERS                         ; PLL
  AptWr     ,00014h,0244Bh   ;PLL_CONTROL                            ; to DSi
  AptWr     ,00014h,0304Bh   ;PLL_CONTROL                            ; timings
  AptWaitSet,00014h,08000h   ;PLL_CONTROL (wait for PLL Lock okay)   ;
  AptClr    ,00014h,00001h   ;PLL_CONTROL (disable PLL Bypass)       ;/
  AptWrMcu  ,02703h,00100h   ;MODE_OUTPUT_WIDTH_A              ;\Size A
  AptWrMcu  ,02705h,000C0h   ;MODE_OUTPUT_HEIGHT_A             ;/ 256x192
  AptWrMcu  ,02707h,00280h   ;MODE_OUTPUT_WIDTH_B              ;\Size B
  AptWrMcu  ,02709h,001E0h   ;MODE_OUTPUT_HEIGHT_B             ;/ 640x480
  AptWrMcu  ,02715h,00001h   ;MODE_SENSOR_ROW_SPEED_A          ;\
  AptWrMcu  ,02719h,0001Ah   ;MODE_SENSOR_FINE_CORRECTION_A    ;
  AptWrMcu  ,0271Bh,0006Bh   ;MODE_SENSOR_FINE_IT_MIN_A        ; Sensor A
  AptWrMcu  ,0271Dh,0006Bh   ;MODE_SENSOR_FINE_IT_MAX_MARGIN_A ;
  AptWrMcu  ,0271Fh,002C0h   ;MODE_SENSOR_FRAME_LENGTH_A       ;
  AptWrMcu  ,02721h,0034Bh   ;MODE_SENSOR_LINE_LENGTH_PCK_A    ;/
  AptWrMcu  ,0A20Bh,  000h   ;AE_MIN_INDEX                     ;\AE min/max
  AptWrMcu  ,0A20Ch,  006h   ;AE_MAX_INDEX                     ;/
  AptWrMcu  ,0272Bh,00001h   ;MODE_SENSOR_ROW_SPEED_B          ;\
  AptWrMcu  ,0272Fh,0001Ah   ;MODE_SENSOR_FINE_CORRECTION_B    ;
  AptWrMcu  ,02731h,0006Bh   ;MODE_SENSOR_FINE_IT_MIN_B        ; Sensor B
  AptWrMcu  ,02733h,0006Bh   ;MODE_SENSOR_FINE_IT_MAX_MARGIN_B ;
  AptWrMcu  ,02735h,002C0h   ;MODE_SENSOR_FRAME_LENGTH_B       ;
  AptWrMcu  ,02737h,0034Bh   ;MODE_SENSOR_LINE_LENGTH_PCK_B    ;/
  AptSet    ,03210h,00008h   ;COLOR_PIPELINE_CONTROL (PGA pixel shading..)
  AptWrMcu  ,0A208h,  000h   ;UNDOC! RESERVED_AE_08
  AptWrMcu  ,0A24Ch,  020h   ;AE_TARGETBUFFERSPEED
  AptWrMcu  ,0A24Fh,  070h   ;AE_BASETARGET
  If Device=7Ah                                          ;\
    AptWrMcu,02717h,00024h   ;MODE_SENSOR_READ_MODE_A    ; Read Mode
    AptWrMcu,0272Dh,00024h   ;MODE_SENSOR_READ_MODE_B    ; with x-flip
```

```
  Else (xflip)                                              ; on internal
    AptWrMcu,02717h,00025h   ;MODE_SENSOR_READ_MODE_A       ; camera
    AptWrMcu,0272Dh,00025h   ;MODE_SENSOR_READ_MODE_B       ;/
  If Device=7Ah                                             ;\
    AptWrMcu,0A202h,  022h   ;AE_WINDOW_POS                 ;
    AptWrMcu,0A203h,  0BBh   ;AE_WINDOW_SIZE                ;
  Else (?)                                                  ;
    AptWrMcu,0A202h,  000h   ;AE_WINDOW_POS                 ;
    AptWrMcu,0A203h,  0FFh   ;AE_WINDOW_SIZE                ;/
  AptSet    ,00016h,00020h   ;CLOCKS_CONTROL (set bit5=1, reserved)
  AptWrMcu  ,0A115h,  072h   ;SEQ_CAP_MODE (was already manipulated above)
  AptWrMcu  ,0A11Fh,  001h   ;SEQ_PREVIEW_1_AWB             ;\
  If Device=7Ah                                             ;
    AptWr   ,0326Ch,00900h   ;APERTURE_PARAMETERS           ;
    AptWrMcu,0AB22h,  001h   ;HG_LL_APCORR1                 ;
  Else (?)                                                  ;
    AptWr   ,0326Ch,01000h   ;APERTURE_PARAMETERS           ;
    AptWrMcu,0AB22h,  002h   ;HG_LL_APCORR1                 ;/
  AptWrMcu     ,0A103h,  006h   ;SEQ_CMD (06h=RefreshMode)
  AptWaitMcuClr,0A103h,  00Fh   ;SEQ_CMD (wait above to become ZERO)
  AptWrMcu     ,0A103h,  005h   ;SEQ_CMD (05h=Refresh)
  AptWaitMcuClr,0A103h,  00Fh   ;SEQ_CMD (wait above to become ZERO)
```

Above does set two Mode/Contexts, 256x192 and 640x480. Yet unknown how to activate the latter one.

aptina_code_list_activate:

```
  AptClr    ,00018h,00001h   ;STANDBY_CONTROL (bit0=0=wakeup)       ;\
  AptWaitClr,00018h,04000h   ;STANDBY_CONTROL (wait for WakeupDone) ; Wakeup
  AptWaitSet,0301Ah,00004h   ;UNDOC_CORE_301A (wait for WakeupDone) ;/
  AptWr     ,03012h,000xxh   ;COARSE_INTEGRATION_TIME (Y Time)
  AptSet    ,0001Ah,00200h   ;RESET_AND_MISC_CONTROL (Parallel On)  ;-Data on
```

Also, don't forget to activate the Camera LED via BPTWL chip (when using the external camera).

aptina_code_list_deactivate:

Before activating another camera: First disable the Parallel output of the old camera (for avoiding collisions on the camera's parallel databus). When not using the camera for longer time, also enter standby mode (for reducing power consumption).

```
  AptClr    ,0001Ah,00200h  ;RESET_AND_MISC_CONTROL (Parallel Off)  ;-Data off
  AptSet    ,00018h,00001h  ;STANDBY_CONTROL (set bit0=1=Standby)   ;\
  AptWaitSet,00018h,04000h  ;STANDBY_CONTROL (wait for StandbyDone) ; Standby
  AptWaitClr,0301Ah,00004h  ;UNDOC_CORE_301A (wait for StandbyDone) ;/
```

References

There aren't any MT9V113 specs released, but info for MT9D113 (a higher resolution variant) does exist: a pdf datasheet, and an xml reference for the I2C registers.

There are several source code files for MT9V113 cameras (different files from different people; for use with linux/android/whatever) including samples for adjusting stuff like contrast and sharpness. However, observe that the source code may need some adjustments: PLL register matched to DSi timings, and, use YUV 8bit parallel databus transfer for DSi.

DSi Aptina Camera Registers: SYSCTL (0000h-0051h)

SYSCTL (0000h-0051h)

```
  0000h   2  CHIP_VERSION_REG       Model ID (2280h=MT9V113 on DSi/3DS) (R)
  0006h  ..  RESERVED_SYSCTL_06     Reserved
  0010h   2  PLL_DIVIDERS           PLL Dividers (def=0366h)
               0-7   PLL M-Divider value (uh, actually a Multiplier?!)
               8-13  PLL N-Divider value
               14-15 Unused (0)
             Because the input clock frequency is unknown, the sensor starts
```

```
          up with the PLL disabled. The PLL takes time to power up. During
          this time, the behavior of its output clock signal is not
          guaranteed. The PLL output frequency is determined by two
          constants, M and N, and the input clock frequency.
            VCO = Fin * 2 * M / (N+1)
            PLL_output_frequency = VCO / (P1+1)
          The PLL can generate a master clock signal whose frequency is up
          to 85 MHz (input clock from 6 MHz through 54 MHz).
0012h   2 PLL_P_DIVIDERS         PLL P Dividers (def=00F5h)
            0-3   P1 (00h..0Fh)
            4-7   Unspecified
            8-11  P3 (00h..0Fh)
            12-13 Division ratio of word clock/clockn from bit_clock (0..3)
            14    Unused (0)
            15    Unspecified
0014h   2 PLL_CONTROL            PLL Control (def=21F9h)
            0     PLL Bypass
            1     PLL Enable
            2-3   Reserved (0..3)
            4-7   Reserved (0..0Fh)
            8     Reset_cntr
            9     Reserved
            10    Reserved
            11    Reserved
            12    Reserved
            13    Reserved
            14    Unused (0)
            15    PLL Lock (R)
0016h   2 CLOCKS_CONTROL         Clocks Control
            0     Reserved
            1     Reserved
            2     Reserved
            3     Reserved
            4     Reserved
            5     Reserved/UNDOC/USED (manipulated by DSi)
            6     Reserved
            7     Reserved
            8     Reserved
            9     clk_clkin_en
            11-12 Reserved
            13    Reserved
            15    Reserved
0018h   2 STANDBY_CONTROL        Standby Control and Status (def=4029h)
            0     Ship (uh?) (0=Enable various regs, 1=Standby)
            1     Reserved
            2     Stop MCU
            3     en_IRQ
            4     Reserved
            5     Reserved
            6-13  Unused (0)
            14    Standby_done (0=WakeupDone, 1=StandbyDone) (R?)
                    (takes MUCH time?)
            15    Reserved (R)
001Ah   2 RESET_AND_MISC_CONTROL Reset and Control (def=0050h) (0-0333h)
            0     Reset SOC I2C
            1     MIPI_TX_Reset
            2     Unused (0)
            3     MIPI_TX_en       (=Serial Data?)
            4     IP_PD_en         (=Parallel Data or what?)
            5     Reserved
            6     Sensor_full_res
            7     Unused (0)
            8     OE_N_Enable
            9     Parallel_enable  (=Parallel Data?)
            10    Unused (0)
```

```
              11    Reserved
              12-15 Unused (0)
  001Ch   2  MCU_BOOT_MODE          MCU Boot Mode
              0     Reset MCU
              1     Reserved
              2     Reserved
              3     Reserved
              4-7   Reserved  (0..0Fh)
              8-15  Reserved  (0..FFh) (R)
  001Eh   2  PAD_SLEW               Pad Slew Control (def=0400h)
              0-2   Parallel Data Output Slew Rate Control (0-7)
              3     Unused (0)
              4-6   GPIO Slew Rate Control (0-7)
              7     Unused (0)
              8-10  PCLK aka PXLCLK Slew Rate Control (0-7)
              11-15 Unused (0)
  0020h  ..  RESERVED_SYSCTL_20     Reserved
  0022h   2  VDD_DIS_COUNTER        VDD_DIS_COUNTER (0..FFFFh, def=0438h)
  0024h   2  GPI_STATUS             GPI_STATUS      (0..000Fh) (R)
  0026h  ..  RESERVED_SYSCTL_26     Reserved
  0028h   2  EN_VDD_DIS_SOFT        EN_VDD_DIS_SOFT (0..0001h, def=0001h)
  0050h  ..  RESERVED_SYSCTL_50     Reserved
```

DSi Aptina Camera Registers: RX_SS, FUSE, XDMA (0100h-099Fh)

RX_SS (0100h-0117h)

```
  0100h  ..  RESERVED_RX_SS_100     Reserved
  0102h   2  TEST_PXL_RED           Test Pixel Red    ;\Default value is 1FFh
  0104h   2  TEST_PXL_G1            Test Pixel Green1 ; for Gray Flat Field
  0106h   2  TEST_PXL_G2            Test Pixel Green2 ; (0..03FFh, def=01FFh)
  0108h   2  TEST_PXL_BLUE          Test Pixel Blue   ;/
  010Ah  ..  RESERVED_RX_SS_10A-116 Reserved
```

FUSE_ROM (0800h-081Fh)

Reserved, unknown purpose, all zero in DSi.

```
  0800h  ..  RESERVED_FUSE_ROM_800-81E  Reserved
```

XDMA (0982h-099Fh)

Access to internal LOGICAL "driver" variables.

```
  0982h  ..  RESERVED_XDMA_982      Reserved
  098Ch   2  MCU_ADDRESS            MCU Address (0000h..FFFFh)
              0-7   driver_variable (0..FFh)
              8-12  driver_id       (0..1Fh) (eg. 3=AWB, 7=MODE, etc.)
              13-14 address space   (0=Physical/RAM/SFR, 1=Logical/Variables)
              15    access_8_bit    (0=16bit, 1=8bit; converted to 16bit)
  0990h 8x2  MCU_DATA_0-7           MCU Data 0..7 (8 x 16bit)
```

For reading, it's best to use "16bit" mode, no matter if reading an 8bit BYTE, or a 16bit MSB,LSB value. The "8bit" mode is converting bytes to 16bit values (MSB=00h, LSB=BYTE), which is a rather contraproductive idiotism; intended for I2C functions that implement only 16bit data transfers, but no 8bit transfers.
Unknown what exactly is mapped at MCU_DATA_0-7 (probably the 16 bytes at MCU_ADDRESS+0..15, probably with direct mapping / ie. without latching a copy of that memory).
MCU_ADDRESS doesn't seem to increment after reading data, however, the i2c index does increase, so one can probably read up to 16 bytes from MCU_DATA_0-7.

DSi Aptina Camera Registers: CORE (3000h-31FFh, 38xxh)

CORE (3000h-31FFh)

```
3000h  ..  RESERVED_CORE_3000        Reserved (same as CHIP_VERSION_REG)
3002h   2  Y_ADDR_START              Y1      ;\Image Position/Size ;def=0004h
3004h   2  X_ADDR_START              X1      ; (up to including    ;def=0004h
3006h   2  Y_ADDR_END                Y2      ; X2,Y2) (0-07FFh)    ;def=04BBh
3008h   2  X_ADDR_END                X2      ;/                    ;def=064Bh
300Ah   2  FRAME_LENGTH_LINES        Y Total ;\Total X/Y Size with ;def=0512h
300Ch   2  LINE_LENGTH_PCK           X Total ;/blanking (0..FFFFh) ;def=0886h
3010h  ..  RESERVED_CORE_3010        Reserved
3012h   2  COARSE_INTEGRATION_TIME   Y Time  ;\Integration Time in ;def=0010h
3014h   2  FINE_INTEGRATION_TIME     X Time  ;/lines/pix (0..FFFFh);def=00F6h
3016h   2  ROW_SPEED                 Row Speed (def=0111h)
             0-2   Pixclk_speed (0..7)
             3     Unused (0)
             4-6   Reserved
             7     Unused (0)
             8-10  Reserved
             11-15 Unused (0)
3018h  ..  RESERVED_CORE_3018-3019   Reserved
301Ah      UNDOC_CORE_301A           Undocumented Status Reg (mask=D7FFh)
             0-1   Unspecified
             2     Undoc/USED (1=WakeupDone) (opposite of 0018h.bit14)
             3-4   Unspecified
             5     Whatever "demo_system, version_reg_write, value=1"
             6-8   Unspecified
             9     Mask_corrupted_frames     (alias of 3022h.bit0)
             10    Unspecified
             11    Unused (0)
             12    Unspecified
             13    Unused (0)
             14    Unspecified
             15    Grouped_parameter_hold    (alias of 3022h.bit8)
301Ch  ..  RESERVED_CORE_301C-3020   Reserved
3022h   2  GROUPED_PARAMETER_HOLD_MASK_CORRUPTED_FRAMES
             0     Mask_corrupted_frames  (alias of Reg 301Ah.bit9)
             1-7   Unused (0)
             8     Grouped_parameter_hold (alias of Reg 301Ah.bit15)
             9-15  Unused (0)
3024h   2  PIXEL_ORDER                Pixel Order (mask=0300h, 0..0300h) (R)
3026h  ..  RESERVED_CORE_3026         Reserved
3028h   2  ANALOGUE_GAIN_CODE_GLOBAL  Analog Global   ;\
302Ah   2  ANALOGUE_GAIN_CODE_GREENR  Analog GreenR   ; Analogue Gain Codes
302Ch   2  ANALOGUE_GAIN_CODE_RED     Analog Red      ; with 3bit fraction
302Eh   2  ANALOGUE_GAIN_CODE_BLUE    Analog Blue     ; (0..007Fh, def=000Bh)
3030h   2  ANALOGUE_GAIN_CODE_GREENB  Analog GreenB   ;/
3032h   2  DIGITAL_GAIN_GREENR        Digital GreenR  ;\Digital Gain with
3034h   2  DIGITAL_GAIN_RED          Digital Red     ; 8bit dummy-fraction
3036h   2  DIGITAL_GAIN_BLUE         Digital Blue    ; (bit8-10=Gain, 0..7)
3038h   2  DIGITAL_GAIN_GREENB       Digital GreenB  ;/(mask=0700h,def=100h)
303Ah  ..  RESERVED_CORE_303A-3C      Reserved
3040h   2  READ_MODE                 Read Mode (0-DEFFh, def=0024h)
             0    horiz_mirror
             1    vert_flip
             2-4  y_odd_inc (0..7)
             5-7  x_odd_inc (0..7)
             8    Unused (0)
             9    low_power
             10   xy_bin_en
             11   x_bin_en
             12   bin_sum (Enable summing mode for binning)
             13   read_mode_y_sumen
             14   Reserved
             15   Reserved
3044h  ..  RESERVED_CORE_3044-3048    Reserved
304Ah   2  OTPM_CONTROL              One-time Programmable Memory? Control
             0    auto_wr_start             ;\
```

```
              1    auto_wr_end (finished) (R) ; automatic write sequence
              2    auto_wr_success (okay) (R) ;/
              3    unspecified
              4    auto_rd_start              ;\
              5    auto_rd_end (finished) (R) ; automatic read sequence
              6    auto_rd_success (okay) (R) ;/
              7-15 Unused (0)
  3050h  ..  RESERVED_CORE_3050-3054    Reserved
  3056h   2  GREEN1_GAIN                Gain Green1             ;\
  3058h   2  BLUE_GAIN                  Gain Blue               ; Gain Values
  305Ah   2  RED_GAIN                   Gain Red                ; (0..0FFFh,
  305Ch   2  GREEN2_GAIN                Gain Green2             ; def=022Ch)
               0-6   Initial Gain (0..7Fh, with 5bit fraction)  ;
               7-8   Analog Gain  (0..3)   (bit8+1)*(bit7+1)*(initial_gain/32)
               9-11  Digital Gain (1..7)                        ;
               12-15 Unused (0)                                 ;/
  305Eh  ..  RESERVED_CORE_305E-31DF    Reserved
  31E0h   2  UNDOC_CORE_31E0            (mask=E003h, 0..8001h, def=0001h) USED!
               Used by DSi (set to 0001h) (reportedly "PIX_DEF_ID")
  31E2h  ..  RESERVED_CORE_31E2-31F9    Reserved
  31FAh   2  UNDOC_CORE_31FA            Whatever (mask=FFFFh, def=CDEFh)
               0-4   Unspecified
               5-11  Whatever "demo_system, version_reg_read, value=3"
               12-15 Unspecified
  31FCh  ..  RESERVED_CORE_305E-31FE    Reserved
```

More CORE (3800h..3803h)

```
  3800h  ..  RESERVED_CORE_3800-3802   Reserved
```

DSi Aptina Camera Registers: SOC1 (3210h-33FDh)

SOC1 Registers (3210h-33FDh)

```
  3210h   2  COLOR_PIPELINE_CONTROL     (mask=05B8h, 0..05B0h, def=01B0h)
               3   Enable PGA pixel shading correction
                     All coefficients and other configuration settings
                     (including other fields in this register) must be set up
                     before enabling shading correction.
               4   Enable 2D aperture correction
               5   Enable color correction
               7   Enable gamma correction
               8   Decimator (1=Enable scale)
               10  Reserved
  3216h  ..  RESERVED_SOC1_3216-321A    Reserved
  321Ch   2  OFIFO_CONTROL_STATUS       Ofifo control status 1 (def=0003h)
               0-3 txfifo_bypass
                     (0=tx_fifo, 1=sensor, 2=sam observe, 3=cpipe format,
                     4=test walking ones cpipe frequency,
                     5=test walking ones sensor frequency,
                     6=RESERVED, 7=test PIXCLK, 8..F=Unspecified)
               4-6 Unused (0)
               7   sensor_bypass (0=cpipe, 1=sensor)
               8   Reserved
               9   Reserved
               10  Reserved
               11  Reserved
               12  Reserved (R)
               13  Reserved (R)
               14  Reserved (R)
               15  Reserved (R)
  321Eh   2  OFIFO_CONTROL_STATUS_2     Ofifo control status 2 (def=0010h)
               0-9   Reserved (0..3FFh)
```

```
            10    Disable PV output clock during blank (1=disable)
            11-15 Reserved (0..1Fh)
3220h  ..  RESERVED_SOC1_3220         Reserved
3222h   2  LOWER_X_BOUND_ZOOM_WINDOW  Lower X  ;def=?      ;\Zoom Window
3224h   2  UPPER_X_BOUND_ZOOM_WINDOW  Upper X  ;def=063Fh  ; Boundaries
3226h   2  LOWER_Y_BOUND_ZOOM_WINDOW  Lower Y  ;def=?      ; (0..07FFh)
3228h   2  UPPER_Y_BOUND_ZOOM_WINDOW  Upper Y  ;def=04AFh  ;/
322Ah   2  UNDOC_SOC1_322A            (mask=0016h, 0..0016h) USED by DSi!
322Ch   2  WEIGHT_HORIZ_DECIMATION    Scaling Weight X ;\Scaling Weight X,Y
322Eh   2  WEIGHT_VERTICAL_DECIMATION Scaling Weight Y ;/(0..0FFFh, def=800h)
323Eh   2  UNDOC_SOC1_323E            (0..FFFFh, def=1A2Dh) (DSi: C22Ch)
3240h   2  UNDOC_SOC1_3240            (0..FFFFh, def=C814h) (DSi: 6214h)
3242h  ..  RESERVED_SOC1_3242         Reserved
3244h   2  UNDOC_SOC1_3244            (mask=03FFh, range=0..00FFh?, def=0310)
3254h  ..  RESERVED_SOC1_3254-326A    Reserved
326Ch   2  APERTURE_PARAMETERS        Aperture Params (0..7FFFh, def=0A08h)
            0-7   2D aperture threshold (knee) (00h-FFh)
            8-10  2D aperture gain (0-7)
            11-13 2D aperture gain's exponent (0-7)
            14    Abs (1=force aperture gain be positive)
            15    Unused (0)
326Eh  ..  RESERVED_SOC1_326E-3276    Reserved
327Ah   2  BLACK_LEVEL_1ST_RED        Offset Red     ;\Offsets subtracted
327Ch   2  BLACK_LEVEL_1ST_GREEN1     Offset Green1  ; from RGB pixels
327Eh   2  BLACK_LEVEL_1ST_GREEN2     Offset Green2  ; (0000-01FFh/03FFh,
3280h   2  BLACK_LEVEL_1ST_BLUE       Offset Blue    ;/def=002Ah)
328Eh   2  THRESH_EDGE_DETECT         Demosaic Edge Threshold (def=000Ch)
3290h   2  TEST_PATTERN               Test Pattern Enable/Width
            0-4   Unused (0)
            5     en_walk_ones_tp  Enable Test Pattern (0=disable, 1=enable)
            6     walk_ones_10     Pattern Width (0=8-bit, 1=10-bit)
            7-15  Unused (0)
329Eh  ..  RESERVED_SOC1_329E-32A0    Reserved
32C0h   2  COLOR_CORR_MATRIX_SCALE_14 Exponents C11..C22 (0-7FFFh, def=3923h)
32C2h   2  COLOR_CORR_MATRIX_SCALE_11 Exponents C23..C33 (0-0FFFh, def=0724h)
32C4h   2  COLOR_CORR_MATRIX_1_2      Elements C11=LSB, C12=MSB   (def=7DCCh)
32C6h   2  COLOR_CORR_MATRIX_3_4      Elements C13=LSB, C21=MSB   (def=2711h)
32C8h   2  COLOR_CORR_MATRIX_5_6      Elements C22=LSB, C23=MSB   (def=62E5h)
32CAh   2  COLOR_CORR_MATRIX_7_8      Elements C31=LSB, C32=MSB   (def=690Dh)
32CCh   2  COLOR_CORR_MATRIX_9        Element C33=LSB, Signs=MSB  (def=2DCDh)
32D4h   2  DIGITAL_GAIN_1_RED         Gain for Red channel    ;\Digital Gain1
32D6h   2  DIGITAL_GAIN_1_GREEN1      Gain for Green1 channel ; (mul 128,
32D8h   2  DIGITAL_GAIN_1_GREEN2      Gain for Green2 channel ; 0000h..03FFh,
32DAh   2  DIGITAL_GAIN_1_BLUE        Gain for Blue channel   ;/def=0080h)
32F4h  ..  RESERVED_SOC1_32F4-332E    Reserved
3330h   2  OUTPUT_FORMAT_TEST         OUTPUT_FORMAT_TEST (0..0FFFh)
            0     Disable Cr channel
            1     Disable Y channel
            2     Disable Cb channel
            3-5   Test ramp output
            6     8+2 bypass
            7     Reserved
            8     Enable Lens Correction Bypass
            9     Reserved
            10    Reserved
            11    Reserved
            12-15 Unused (0)
3332h  ..  RESERVED_SOC1_3332-334A    Reserved
337Ch   2  YUV_YCBCR_CONTROL          YUV_YCBCR_CONTROL (0..000Fh, def=0006h)
            0     Mult_y_uv (normalize Y in 16-235; U and V in 16-240)
            1     Coefficient control
            2     Add 128 to U and V
            3     Clip Y in 16-235; U and V in 16-240
            4-15  Unused (0)
337Eh   2  Y_RGB_OFFSET               Y_RGB Offset
```

```
              0-7   Reserved (0..FFh)
              8-15  Y offset (0..FFh)
  33E6h  ..  RESERVED_SOC1_33E6-33EE    Reserved
  33F4h   2  KERNEL_CONFIG              Kernel Config (0..01FFh, def=0003h)
              0     Defect correction (DC) enable
              1     Reserved
              2     Reserved
              3     Noise reduction (NR) enable
              4     Reserved
              5     Reserved
              6     Reserved
              7     Reserved
              8     Reserved
  33F6h  ..  RESERVED_SOC1_33F6-33FC    Reserved
```

DSi Aptina Camera Registers: SOC2 (3400h-3729h)

SOC2 (3400h-3729h)

```
  3400h   2  MIPI_CONTROL              MIPI_Control (def=782Eh)
              0     MIPI restart enable
              1     MIPI standby
              2     Continuous MIPI clock
              3     Frame boundary sync bit (R)
              4     Wait until eof to react to standby
              5     Reserved
              6-8   MIPI channel number
              9     Unused (0) or Reserved (REV3)
              10-15 Data Type (1Eh=YUV422_8bit, 20h=RGB444, 21h=RGB555,
                      22h=RGB565, 2Ah=RAW8, 2Bh=RAW10)
  3402h   2  MIPI_STATUS               MIPI_Status (def=0011h)
              0     MIPI in standby (R)
              1-3   Unused (0)
              4     MIPI aka MIPICCP idle (R)
              5     MIPI ready to receive data (R)
              6-8   Unused (0)
              9     Reserved (R)
              10    Reserved (R)
              11    Reserved
              12    Reserved
              13-15 Unused (0)
  3404h   2  CUSTOM_SHORT_PKT          MIPI_Custom_Short_Packet (0000h-3F00h)
              0-5   Unused (0)
              6     frame_cnt_reset (sent in frame start/end short packets)
              7     frame_cnt_en (Insert frame counter value in WC field)
              8-10  custom_short_packet_data_type
              11    custom_short_packet_request
              12    custom_short_packet_frame_sync
              13    custom_short_packet_reset (R)
              14-15 Unused (0)
  3408h   2  LINE_BYTE_CNT             MIPI line byte count (def=0C80h)
  340Ch   2  CUSTOM_SHORT_PKT_WC       WC field of a custom short packet
  340Eh  ..  RESERVED_SOC2_340E-341A   Reserved
  3580h   2  AE_ZONE_X                 AE Window/Zone X (def=1300h)
              0-7:  ae_zone_x_start (00h..FFh) (div8)          ;for WINDOW
              8-15: ae_zone_x_width (00h..FFh) (div8, minus 1) ;for each ZONE
  3582h   2  AE_ZONE_Y                 AE Window/Zone Y (def=0E00h)
              0-7:  ae_zone_y_start (00h..FFh) (div8)          ;for WINDOW
              8-15: ae_zone_y_width (00h..FFh) (div8, minus 1) ;for each ZONE
  3584h   2  AE_WINDOW_SIZE_LO         LSBs ;\Size of each AE zone in pixels
  3586h   2  AE_WINDOW_SIZE_HI         MSBs ;/(0..0001FFFFh, def=000x4B00h ?)
  3588h  ..  RESERVED_SOC2_3588-35AE   Reserved
  35B0h      UNDOC_SOC2_35B0           (mask=FFFFh, 0..FFFFh, def=05FAh) USED!
```

```
35B2h  ..  RESERVED_SOC2_35B2-3602   Reserved
3604h  20  R_GAMMA_CURVE_KNEES_0-18  Red Gamma Curve Knees 0..18   (1B00h,..)
3618h  20  G_GAMMA_CURVE_KNEES_0-18  Green Gamma Curve Knees 0..18 (1B00h,..)
362Ch  20  B_GAMMA_CURVE_KNEES_0-18  Blue Gamma Curve Knees 0..18  (1B00h,..)
             Above 20-byte knees consist of ten 16bit values (Knee0 in LSB)
             Due to the 16bit-big-endian format, the byte-order is:
             Knee1,Knee0,Knee3,Knee2,...,Knee17,Knee16,UNUSED,Knee18
3640h  ..  RESERVED_SOC2_3640        Reserved
3642h   2  POLY_ORIGIN_R             Center Row    (max 07FFh, def=025Ch)
3644h   2  POLY_ORIGIN_C             Center Column (max 07FFh, def=0324h)
3646h  ..  RESERVED_SOC2_3646-364C   Reserved
364Eh 5x2  P_GR_P0Q0-4               P0Q for Green1  ;\P0 Coefficients
3658h 5x2  P_RD_P0Q0-4               P0Q for Red     ; (5 x float16 each)
3662h 5x2  P_BL_P0Q0-4               P0Q for Blue    ; (0010h,... each)
366Ch 5x2  P_GB_P0Q0-4               P0Q for Green2  ;/
3676h 5x2  P_GR_P1Q0-4               P1Q for Green1  ;\
3680h 5x2  P_RD_P1Q0-4               P1Q for Red     ; P1 Coefficients
368Ah 5x2  P_BL_P1Q0-4               P1Q for Blue    ; (5 x float16 each)
3694h 5x2  P_GB_P1Q0-4               P1Q for Green2  ;/
369Eh 5x2  P_GR_P2Q0-4               P2Q for Green1  ;\
36A8h 5x2  P_RD_P2Q0-4               P2Q for Red     ; P2 Coefficients
36B2h 5x2  P_BL_P2Q0-4               P2Q for Blue    ; (5 x float16 each)
36BCh 5x2  P_GB_P2Q0-4               P2Q for Green2  ;/
36C6h 5x2  P_GR_P3Q0-4               P3Q for Green1  ;\
36D0h 5x2  P_RD_P3Q0-4               P3Q for Red     ; P3 Coefficients
36DAh 5x2  P_BL_P3Q0-4               P3Q for Blue    ; (5 x float16 each)
36E4h 5x2  P_GB_P3Q0-4               P3Q for Green2  ;/
36EEh 5x2  P_GR_P4Q0-4               P4Q for Green1  ;\
36F8h 5x2  P_RD_P4Q0-4               P4Q for Red     ; P4 Coefficients
3702h 5x2  P_BL_P4Q0-4               P4Q for Blue    ; (5 x float16 each)
370Ch 5x2  P_GB_P4Q0-4               P4Q for Green2  ;/
3716h  ..  RESERVED_SOC2_3716-3278   Reserved
```

DSi Aptina Camera Variables: RAM/SFR/MON (GPIO/Monitor) (MCU:0000h-20xxh)

Internal RAM (MCU:0000h..0xxxh)

Internal RAM is reserved for whatever internal purposes (probably including for storing the 'logical variables' at MCU:2xxxh at some physical memory locations at MCU:0xxxh). However, some of those undocumented reserved RAM cells are manipulated by DSi games:

```
02F0h   2  UNDOC_RAM_02F0   (set to 0000h by DSi games)
02F2h   2  UNDOC_RAM_02F2   (set to 0210h by DSi games)
02F4h   2  UNDOC_RAM_02F4   (set to 001Ah by DSi games)
```

Exact RAM Size is unknown (around 2Kbyte or so)? Some Aptina chips do also contain some sort of User RAM (at 0800h or so) for unknown purpose (just general purpose storage maybe). Unknown if the DSi chips are having any such User RAM.

Special Function Registers SFR (MCU:1040h..10FEh)

```
1040h  ..  RESERVED_SFR_1040-1050    Reserved
1060h  ..  RESERVED_SFR_1060-1066    Reserved (REV3)
1070h   2  GPIO_DATA                 GPIO Data  (0..1E00h)
             0-8   Unused (0)
             9-12  gpio_3_0_data
             13-15 Unused (0)
1072h   2  RESERVED_SFR_1072         Reserved
1074h   2  GPIO_OUTPUT_SET           GPIO Set   (0..0C00h/1E00h?) (W)
             0-8   Unused (0)
             9-12  gpio_3_0_output_toggle (uh, toggle or set?)
             13-15 Unused (0)
1076h   2  GPIO_OUTPUT_CLEAR         GPIO Clear (0..0C00h/1E00h?) (W)
```

```
              0-8   Unused (0)
              9-12  gpio_3_0_output_clear
              13-15 Unused (0)
  1078h   2  GPIO_DIR                 GPIO Direction (0..1E00h, def=1E00h)
              0-8   Unused (0)
              9     gpio_0_dir (0=Output, 1=Input) ;(LSB0 of 10bit Output)
              10    gpio_1_dir (0=Output, 1=Input) ;(LSB1 of 10bit Output)
              11    gpio_2_dir (0=Output, 1=Input) ;(Flash/Shutter Pulse)
              12    gpio_3_dir (0=Output, 1=Input) ;(OE_BAR for Databus)
              13-15 Unused (0)
  107Ah  ..  RESERVED_SFR_107A-10FD    Reserved
```

Monitor Variables MON (MCU:2000h..2025h)

```
  2000h   5  RESERVED_MON_00-04        Reserved
  2005h   1  MON_CMD                   Monitor Command (0..FFh)
  2006h   2  MON_ARG1                  Monitor First Argument (0..FFFFh)
  2008h  ..  RESERVED_MON_08-22        Reserved
  2024h   2  MON_PATCH_ID_0            Monitor First Patch (0..FFFFh) (REV1)
              0-7   mon_patch_0_version (00h-0Fh)
                      The version number of the first patch (R)
              8-15  mon_patch_0_number  (00h-0Fh)
                      Identifies which patch the first patch is (R)
  2024h   1  MON_PATCH_ID_0            (mask=FFh) (R) ;\unlike above  (REV3)
  2025h   1  MON_PATCH_ID_1            (0..FF)        ;/REV1 specs    (REV3)
  2026h   1  MON_PATCH_ID_2            (0..FF)                        (REV3)
  2027h   1  RESERVED_MON_27           Reserved                       (REV3)
```

DSi Aptina Camera Variables: SEQ (Sequencer) (MCU:21xxh)

Sequencer Variables SEQ (MCU:2100h..215Ah)

```
  2100h  ..  RESERVED_SEQ_00           Reserved
  2102h   1  SEQ_MODE                  SEQ Mode (enables "drivers") (def=0Fh)
              0    Enable AE   (ID=2)
              1    Enable FD   (ID=4)
              2    Enable AWB  (ID=3)
              3    Enable HG   (ID=11)
              4-7  Unspecified
  2103h   1  SEQ_CMD                   SEQ Cmd   (0..FFh, def=01h)
              0-7  Cmd   (0=Run, 1=Preview, 2=Capture, 3=Standby,
                         4=Lock, 5=Refresh, 6=Refresh Mode)
  2104h   1  SEQ_STATE                 SEQ State (0..FFh)
              0-7  State (0=Run, 1=ToPreview, 2=Enter, 3=Preview
                          4=Leave, 5=ToCapture, 6=Enter, 7=Capture,
                          8=Leave, 9=Standby)
  2105h  ..  RESERVED_SEQ_05           Reserved
  2106h   1  SEQ_FLASHTYPE             Type of flash to be used
              0-6 Flash Type (0=None, 1=LED, 2=Xenon, 3=XenonBurst)
              7   Set flash to LOCK mode (0=Normal, 1=LOCK mode)
  2107h  ..  RESERVED_SEQ_07-08        Reserved
  2109h   1  SEQ_AE_FASTBUFF           AE Fast Buff  (0..FFh, def=10h)
  210Ah   1  SEQ_AE_FASTSTEP           AE Fast Step  (0..FFh, def=02h)
  210Bh   1  SEQ_AWB_CONTBUFF          AWB Cont Buff (0..FFh, def=08h)
  210Ch   1  SEQ_AWB_CONTSTEP          AWB Cont Step (0..FFh, def=02h)
  210Dh  ..  RESERVED_SEQ_0D-10        Reserved
  2111h   1  SEQ_OPTIONS               SEQ Options (0..FFh, def=08h)
              0  Reserved
              1  Reserved
              2  Reserved
              3  seq_crop_win_ae, Use crop window for AE statistics
              4  seq_crop_win_awb, Use crop window for AWB statistics
              7  Reserved
  2112h  ..  RESERVED_SEQ_12           Reserved
```

```
2113h   2  SEQ_FLASH_TH              SEQ Flash TH (0..FFFFh)
2115h   1  SEQ_CAP_MODE              Capture mode (in Capture state only)
             0  Xenon Flash (Still Only)
             1  Video
             2  Turn Flash off before last frame in capture state
             4  Video AE on
             5  Video AWB on
             6  Video HG on
2116h   1  SEQ_CAP_NUMFRAMES         Num still frames captured (0..FFh,def=3)
2117h   1  SEQ_PREVIEW_0_AE          Preview 0 AE (PREVIEW ENTER)     ;\
             0-3  AE (0=Off, 1=Fast, 2=Manual, 3=Continuous, 4=MDR)   ;
             4-7  Unspecified (0..5) (0..0Fh for PREVIEW_2/3)         ; Pre-
2118h   1  SEQ_PREVIEW_0_FD          Preview 0 FD (PREVIEW ENTER)     ; view
             0-7  FD (0=Off, 1=Continuous, 2=Manual)                  ; 0
2119h   1  SEQ_PREVIEW_0_AWB         Preview 0 AWB (PREVIEW ENTER)    ;
             0-7  AWB (0=Off, 1=On)                                   ; PRE-
211Ah   1  SEQ_PREVIEW_0_HG          Preview 0 HG (PREVIEW ENTER)     ; VIEW
             0-7  HG (0=Off, 1=Fast, 2=Manual, 3=Continuous)          ; ENTER
211Bh   1  SEQ_PREVIEW_0_FLASH       Flash Config (0..FFh)            ;
             0-6 Flash (0=Off,1=On,2=Locked,3=AutoEvaluate,7=UserDef) ;
             7   Reserved                                            ;
211Ch   1  SEQ_PREVIEW_0_SKIPFRAME   Skipframe State Config (def=40h) ;
             0-3 Unspecified                                         ;
             4   Unspecified (except PREVIEW_2: Reserved)            ;
             5   Skip_led_on                                         ;
             6   Skip_state (0=No skip state, 1=Skip state)          ;
             7   Turn_off_fen                                        ;/
211Dh   1  SEQ_PREVIEW_1_AE            ;\                           def=01h
211Eh   1  SEQ_PREVIEW_1_FD            ; Preview 1 (PREVIEW)        def=01h
211Fh   1  SEQ_PREVIEW_1_AWB           ; (same as Preview 0, but    def=01h
2120h   1  SEQ_PREVIEW_1_HG            ; without AE=MDR,            def=01h
2121h   1  SEQ_PREVIEW_1_FLASH         ; without HG=Manual/Continous)
2122h   1  SEQ_PREVIEW_1_SKIPFRAME     ;/                           def=N/A
2123h   1  SEQ_PREVIEW_2_AE            ;\
2124h   1  SEQ_PREVIEW_2_FD            ; Preview 2 (PREVIEW LEAVE)
2125h   1  SEQ_PREVIEW_2_AWB           ; (same as Preview 0, but
2126h   1  SEQ_PREVIEW_2_HG            ; without HG=Manual/Continous)
2127h   1  SEQ_PREVIEW_2_FLASH         ;
2128h   1  SEQ_PREVIEW_2_SKIPFRAME     ;/
2129h   1  SEQ_PREVIEW_3_AE            ;\
212Ah   1  SEQ_PREVIEW_3_FD            ; Preview 3 (CAPTURE ENTER)
212Bh   1  SEQ_PREVIEW_3_AWB           ; (same as Preview 0)
212Ch   1  SEQ_PREVIEW_3_HG            ;
212Dh   1  SEQ_PREVIEW_3_FLASH         ;
212Eh   1  SEQ_PREVIEW_3_SKIPFRAME     ;/
212Fh  ..  RESERVED_SEQ_2F-33         Reserved
2134h   1  UNDOC_SEQ_34               (0..FFh)
2135h  ..  RESERVED_SEQ_35-44         Reserved
2145h   2  UNDOC_SEQ_45               (0..FFFFh)
2147h  ..  RESERVED_SEQ_47-59         Reserved
```

DSi Aptina Camera Variables: AE (Auto Exposure) (MCU:22xxh)

Auto Exposure Variables AE (MCU:2200h-2261h)

```
2200h  ..  RESERVED_AE_00             Reserved
2202h   1  AE_WINDOW_POS              AE Window Position Y0 and X0
             0-3  X0 (in units of 1/16th of frame width)  (0..0Fh)
             4-7  Y0 (in units of 1/16th of frame height) (0..0Fh)
2203h   1  AE_WINDOW_SIZE             AE Window Height and Width (def=FFh)
             0-3  Width  (units of 1/16th of frame width, minus 1)  (0..0Fh)
             4-7  Height (units of 1/16th of frame height, minus 1) (0..0Fh)
2204h  ..  RESERVED_AE_04             Reserved
```

```
2206h   1  AE_TARGET                AE Target Brightness (0..FFh, def=32h)
2207h   1  AE_GATE                  AE Sensitivity       (0..FFh, def=04h)
2208h  ..  UNDOC_AE_08               (0..FFh, def=02h)
2209h  ..  RESERVED_AE_09-0A         Reserved
220Bh   1  AE_MIN_INDEX              Min                        (0-FFh)
220Ch   1  AE_MAX_INDEX              Max allowed zone number  (0-FFh,def=18h)
220Dh   1  AE_MIN_VIRTGAIN           Min allowed virtual gain (0-FFh,def=10h)
220Eh   1  AE_MAX_VIRTGAIN           Max allowed virtual gain (0-FFh,def=80h)
220Fh  ..  RESERVED_AE_0F-11         Reserved
2212h   2  AE_MAX_DGAIN_AE1         Max digital gain pre-LC  (def=8000h)
2214h  ..  RESERVED_AE_14-16         Reserved
2217h   1  AE_STATUS                AE Status
             0   AE_at_limit (1=AE reached limit)
             1   R9_changed  (1=Need to skip frame)
             2   Ready       (0=AE not ready, 1=AE ready)
             3-7 Unused (0)
2218h   1  AE_CURRENT_Y             Last measured luma   (0-FFh,def=4Bh) (R)
2219h   2  AE_R12                   Curr shutter delay       (def=0279h) (R)
221Bh   1  AE_INDEX                 Curr zone integration time (def=04h) (R)
221Ch   1  AE_VIRTGAIN              Curr virtual gain    (0-FFh,def=10h) (R)
221Dh  ..  RESERVED_AE_1D-1E         Reserved
221Fh   2  AE_DGAIN_AE1              Current digital gain pre-LC (def=0080h)
2221h  ..  RESERVED_AE_21            Reserved
2222h   2  AE_R9                     Current R9:0 value (0-FFFFh, def=0010h)
2224h  ..  RESERVED_AE_24-2C         Reserved
222Dh   2  AE_R9_STEP                Integration time per zone   (def=009Dh)
222Fh  ..  RESERVED_AE_2F-49         Reserved
224Ah   1  AE_TARGETMIN              Min value for target  (0..FFh, def=32h)
224Bh   1  AE_TARGETMAX              Max value for target  (0..FFh, def=96h)
224Ch   1  AE_TARGETBUFFERSPEED      Target Buffer Speed   (0..FFh, def=0Ch)
224Dh  ..  RESERVED_AE_4D            Reserved
224Fh   1  AE_BASETARGET             Target Base           (0..FFh, def=36h)
2250h  ..  RESERVED_AE_50-61         Reserved
2262h  ..  RESERVED_AE_62-64         Reserved (REV3)
```

DSi Aptina Camera Variables: AWB (Auto White Balance) (MCU:23xxh)

Auto White Balance Variables AWB (MCU:2300h..236Eh)

```
2300h  ..  RESERVED_AWB_00           Reserved
2302h   1  AWB_WINDOW_POS            AWB Window Position Y0 and X0
             0-3  X0 (in units of 1/16th of frame width)  (0..0Fh)
             4-7  Y0 (in units of 1/16th of frame height) (0..0Fh)
2303h   1  AWB_WINDOW_SIZE           AWB Window Size (def=EFh)
             0-3  Width  (units of 1/16th of frame width, minus 1)  (0..0Fh)
             4-7  Height (units of 1/16th of frame height, minus 1) (0..0Fh)
2304h  ..  RESERVED_AWB_04           Reserved
2306h 3x2  AWB_CCM_L_0-2             Left CCM K11,K12,K13 (0180h,FF00h,0080h)
230Ch 3x2  AWB_CCM_L_3-5             Left CCM K21,K22,K23 (FF66h,0180h,FFEEh)
2312h 3x2  AWB_CCM_L_6-8             Left CCM K31,K32,K33 (FFCDh,FECDh,019Ah)
2318h   2  AWB_CCM_L_9               Left CCM Red/Green gain  (0020h)
231Ah   2  AWB_CCM_L_10              Left CCM Blue/Green gain (0033h)
231Ch 3x2  AWB_CCM_RL_0-2            DeltaCCM D11,D12,D13 (0100h,FF9Ah,xxxxh)
2322h 3x2  AWB_CCM_RL_3-5            DeltaCCM D21,D22,D23 (004Dh,FFCDh,FFB8h)
2328h 3x2  AWB_CCM_RL_6-8            DeltaCCM D31,D32,D33 (004Dh,0080h,FF66h)
232Eh   2  AWB_CCM_RL_9              DeltaCCM Red/Green gain  (0008h)
2330h   2  AWB_CCM_RL_10             DeltaCCM Blue/Green gain (FFF7h)
2332h 3x2  AWB_CCM_0-2               Curr CCM C11,C12,C13 (01BAh,FF5Bh,FFF1h)
2338h 3x2  AWB_CCM_3-5               Curr CCM C21,C22,C23 (FFC7h,01B9h,FF87h)
233Eh 3x2  AWB_CCM_6-8               Curr CCM C31,C32,C33 (FFF9h,FF32h,01DCh)
2344h   2  AWB_CCM_9                 Curr CCM Red/Green gain  (003Ch)
```

```
2346h   2  AWB_CCM_10                Curr CCM Blue/Green gain (002Bh)
2348h   1  AWB_GAIN_BUFFER_SPEED     Gain Speed (1-20h, def=08h, 20h=fastest)
2349h   1  AWB_JUMP_DIVISOR          Jump Divisor (1-FFh, def=02h, 1=fastest)
234Ah   1  AWB_GAIN_MIN              Min AWB Red     (def=59h) ;\Digital Gain
234Bh   1  AWB_GAIN_MAX              Max allowed Red (def=B6h) ; Min/max
234Ch   1  AWB_GAINMIN_B             Min AWB         (def=59h) ; (0..FFh)
234Dh   1  AWB_GAINMAX_B             Max allowed     (def=A6h) ;/
234Eh   1  AWB_GAIN_R                Current R digital gain    ;\Current Gain
234Fh   1  AWB_GAIN_G                Current G digital gain    ; (0..FFh,
2350h   1  AWB_GAIN_B                Current B digital gain    ;/def=80h)
2351h   1  AWB_CCM_POSITION_MIN      Min/Left  (def=?)   ;\(range 0..FFh,
2352h   1  AWB_CCM_POSITION_MAX      Max/Right (def=7Fh) ; 00h=incandescent,
2353h   1  AWB_CCM_POSITION          Position  (def=40h) ;/7Fh=daylight)
2354h   1  AWB_SATURATION            Saturation (0..FFh, def=80h, 80h=100%)
2355h   1  AWB_MODE                  Misc control for AWB (0..FFh)
             0  Steady         (1=AWB is done)
             1  Limits Reached (1=AWB limit is reached)
             2  Reserved
             3  Reserved
             4  Reserved
             5  Force_unit_dgains
             6  NormCCM_off
2356h   2  AWB_GAINR_BUF             Time-buffered R gain (0..FFFFh)
2358h   2  AWB_GAINB_BUF             Time-buffered B gain (0..FFFFh)
235Ah  ..  RESERVED_AWB_5A-5C        Reserved
235Dh   1  AWB_STEADY_BGAIN_OUT_MIN  (0-FF, def=78h)
235Eh   1  AWB_STEADY_BGAIN_OUT_MAX  (0-FF, def=86h)
235Fh   1  AWB_STEADY_BGAIN_IN_MIN   (0-FF, def=7Eh)
2360h   1  AWB_STEADY_BGAIN_IN_MAX   (0-FF, def=82h)
2361h   2  UNDOC_AWB_61              (0..FFFFh, def=0040h)
2363h   1  AWB_TG_MIN0               True Gray minimum (0..FFh, def=D2h)
2364h   1  AWB_TG_MAX0               True Gray maximum (0..FFh, def=F6h)
2365h   1  AWB_X0                    (0-FFh, def=10h)
2366h   1  AWB_KR_L                  (0-FFh, def=80h)
2367h   1  AWB_KG_L                  (0-FFh, def=80h)
2368h   1  AWB_KB_L                  (0-FFh, def=80h)
2369h   1  AWB_KR_R                  (0-FFh, def=80h)
236Ah   1  AWB_KG_R                  (0-FFh, def=80h)
236Bh   1  AWB_KB_R                  (0-FFh, def=80h)
236Ch  ..  RESERVED_AWB_6C-6E        Reserved
```

DSi Aptina Camera Variables: FD (Anti-Flicker) (MCU:24xxh)

Anti-Flicker Variables FD (MCU:2400h..247Bh)

```
2400h  ..  RESERVED_FD_00            Reserved
2402h   1  FD_WINDOW_POSH            Window Pos H (0..FFh, def=1Dh)
             0-3  Width (in units of 1/16th of frame width, minus 1) (0..0Fh)
             4-7  X0 (=position/origin or so?) (0..0Fh)
2403h   1  FD_WINDOW_HEIGHT          FlickerMeasurementWindowHeight (def=04h)
             0-5  Flicker measurement window height in rows (0..3Fh)
             6-7  Unspecified
2404h   1  FD_MODE                   Flicked Detection switches/indicators
             0-3  Reserved      (0..0Fh) (R)
             4    Debug_mode    (0=Disable, 1=Enable single period mode)
             5    Curr Flicker State (0=60Hz, 1=50Hz) (R)
             6    Curr Settings (0=60Hz, 1=50Hz)
             7    Manual Mode   (0=Disable, 1=Enable)
2405h  ..  RESERVED_FD_05-07         Reserved
2408h   1  FD_SEARCH_F1_50          Search F1 50Hz  (0..FFh, def=33h)
2409h   1  FD_SEARCH_F2_50          Search F2 50Hz  (0..FFh, def=35h)
240Ah   1  FD_SEARCH_F1_60          Search F1 60Hz  (0..FFh, def=29h)
240Bh   1  FD_SEARCH_F2_60          Search F2 60Hz  (0..FFh, def=2Bh)
```

```
240Ch   1  UNDOC_FD_0C              (0..FFh)
240Dh   1  FD_STAT_MIN              Stat Min (0..FFh, def=03h)
240Eh   1  FD_STAT_MAX              Stat Max (0..FFh, def=05h)
240Fh  ..  RESERVED_FD_0F           Reserved
2410h   1  FD_MIN_AMPLITUDE         Ignore Signals below Min (0..FFh, def=5)
2411h   2  FD_R9_STEP_F60_A         60HzA (def=0D4h) ;\Minimal Shutter Width
2413h   2  FD_R9_STEP_F50_A         50HzA (def=103h) ; Steps for 60Hz/50H AC
2415h   2  FD_R9_STEP_F60_B         60HzB (def=09Dh) ; in Context A/B
2417h   2  FD_R9_STEP_F50_B         50HzB (def=0B8h) ;/(0..FFFFh)
2419h  ..  RESERVED_FD_19-7B        Reserved
```

DSi Aptina Camera Variables: MODE (Mode/Context) (MCU:27xxh)

Mode Variables MODE (MCU:2700h..2768h)

```
2700h  ..  RESERVED_MODE_00-02              Reserved
2703h   2  MODE_OUTPUT_WIDTH_A        (CX)  (0..FFFFh, def=0320h)  ;\Size A
2705h   2  MODE_OUTPUT_HEIGHT_A       (CY)  (0..FFFFh, def=0258h)  ;/
2707h   2  MODE_OUTPUT_WIDTH_B              (0..FFFFh, def=0640h)  ;\Size B
2709h   2  MODE_OUTPUT_HEIGHT_B             (0..FFFFh, def=04B0h)  ;/
270Bh   1  MODE_A_MIPI_VC                   (0..07h) (REV3)        ;-Mipi A
270Ch   1  MODE_B_MIPI_VC                   (0..07h) (REV3)        ;-Mipi B
270Dh   2  MODE_SENSOR_ROW_START_A    (Y1)  (0..FFFFh)             ;\
270Fh   2  MODE_SENSOR_COL_START_A    (X1)  (0..FFFFh)             ;
2711h   2  MODE_SENSOR_ROW_END_A      (Y2)  (0..FFFFh, def=040Dh)  ;
2713h   2  MODE_SENSOR_COL_END_A      (X2)  (0..FFFFh, def=050Dh)  ; Sensor
2715h   2  MODE_SENSOR_ROW_SPEED_A          (0..0777h, def=0112h)  ; A
             0-2:  pixclk_speed (0..7)                             ;
                     1ADC: Pclk = 2 mclks * bits[0:2]              ;
                     2ADC: bits[0:2]                               ;
             4-6:  Reserved (0..7)                                 ;
             8-10: Reserved (0..7)                                 ;
2717h   2  MODE_SENSOR_READ_MODE_A          (0..FFFFh, def=046Ch)  ;
             0:   horiz_mirror                                     ;
             1:   vert_flip                                        ;
             2-4: y_odd_inc (0..7)                                 ;
             5-7: x_odd_inc (0..7)                                 ;
             9:   low_power                                        ;
             10:  xy_bin_en                                        ;
             11:  x_bin_en                                         ;
2719h   2  MODE_SENSOR_FINE_CORRECTION_A    (0..FFFFh, def=007Bh)  ;
271Bh   2  MODE_SENSOR_FINE_IT_MIN_A        (0..FFFFh, def=0408h)  ;
271Dh   2  MODE_SENSOR_FINE_IT_MAX_MARGIN_A (0..FFFFh, def=00ABh)  ;
271Fh   2  MODE_SENSOR_FRAME_LENGTH_A       (0..FFFFh, def=0293h)  ;
2721h   2  MODE_SENSOR_LINE_LENGTH_PCK_A    (0..FFFFh, def=07D0h)  ;/
2723h   2  MODE_SENSOR_ROW_START_B          (0..FFFFh, def=0004h)  ;\
2725h   2  MODE_SENSOR_COL_START_B          (0..FFFFh, def=0004h)  ; Sensor
2727h   2  MODE_SENSOR_ROW_END_B            (0..FFFFh, def=040Bh)  ; B
2729h   2  MODE_SENSOR_COL_END_B            (0..FFFFh, def=050Bh)  ;
272Bh   2  MODE_SENSOR_ROW_SPEED_B          (0..0777h, def=0111h)  ; (same
272Dh   2  MODE_SENSOR_READ_MODE_B          (0..FFFFh, def=0024h)  ; as
272Fh   2  MODE_SENSOR_FINE_CORRECTION_B    (0..FFFFh, def=00A4h)  ; Sensor
2731h   2  MODE_SENSOR_FINE_IT_MIN_B        (0..FFFFh, def=0408h)  ; A, see
2733h   2  MODE_SENSOR_FINE_IT_MAX_MARGIN_B (0..FFFFh, def=00A4h)  ; there)
2735h   2  MODE_SENSOR_FRAME_LENGTH_B       (0..FFFFh, def=04EDh)  ;
2737h   2  MODE_SENSOR_LINE_LENGTH_PCK_B    (0..FFFFh, def=0D06h)  ;/
2739h   2  MODE_CROP_X0_A                   (0..FFFFh)             ;\
273Bh   2  MODE_CROP_X1_A                   (0..FFFFh, def=031Fh)  ; Crop A
273Dh   2  MODE_CROP_Y0_A                   (0..FFFFh)             ;
273Fh   2  MODE_CROP_Y1_A                   (0..FFFFh, def=0257h)  ;/
2741h  ..  RESERVED_MODE_41-45              Reserved
2747h   2  MODE_CROP_X0_B                   (0..FFFFh)             ;\
2749h   2  MODE_CROP_X1_B                   (0..FFFFh, def=063Fh)  ; Crop B
```

```
274Bh   2  MODE_CROP_Y0_B                  (0..FFFFh)             ;
274Dh   2  MODE_CROP_Y1_B                  (0..FFFFh, def=04AFh)  ;/
274Fh  ..  RESERVED_MODE_4F-53             Reserved
2755h   2  MODE_OUTPUT_FORMAT_A            Format A (0..FFFFh     ;\
2757h   2  MODE_OUTPUT_FORMAT_B            Format B (0..FFFFh     ;
             0     swap_channels (swap Cb/Cr in YUV and R/B in RGB);
             1     swap_chrominance_luma                          ; Format
             2     bayer_out (Progressive Bayer)                  ; A/B
             3     monochrome (0..1)                              ;
             4     Reserved                                       ;
             5     output_mode (0=YUV, 1=RGB)                     ;
             6-7   RGB Format (0=565, 1=555, 2=444xh, 3:x444h)    ;
             8     Processed Bayer (0..1)                         ;
             9     Invert out_clk (0..1) (REV3)                   ;
             10-15 Unspecified                                    ;/
2759h   2  MODE_SPEC_EFFECTS_A             Effects A  (def=6440h) ;\
275Bh   2  MODE_SPEC_EFFECTS_B             Effects B  (def=6440h) ;
             0-2  Selection (1=Mono, 2=Sepia, 3=Negative,         ; Effects
                    4=Solarization, 5=Solarization w/ UV)         ; A/B
             3-5  Dither_bitwidth                                 ;
             6    Dither_luma                                     ;
             8-15 Solarization Threshold (0..7 for diff effects)  ;/
275Dh   1  MODE_Y_RGB_OFFSET_A             Offset A (00h..FFh)    ;\Offset
275Eh   1  MODE_Y_RGB_OFFSET_B             Offset B (00h..FFh)    ;/A/B
275Fh   2  MODE_COMMON_MODE_SETTINGS_BRIGHT_COLOR_KILL             ;\
             Shadow register for 35A4h in SOC2                    ;
             0-2   Color kill saturation point (0..7)             ; Kill
             3-5   Bright color kill gain      (0..7)             ; Bright
             6-8   Bright color kill threshold (0..7)             ;
             9     Signal_ctrl (1=use luma as min/max value)      ;
             10    en_kl       (1=enable bright color kill)       ;
             11-15 Unspecified                                    ;/
2761h   2  MODE_COMMON_MODE_SETTINGS_DARK_COLOR_KILL               ;\
             Shadow register for 35A2h in SOC2                    ;
             0-2   Dark color kill gain      (0..7)               ; Kill
             3-5   Dark color kill threshold (0..7)               ; Dark
             6     Signal_ctrl (1=use luma as min/max value)      ;
             7     en_dark_kl  (1=enable dark color kill)         ;
             8-15  Unspecified                                    ;/
2763h   2  MODE_COMMON_MODE_SETTINGS_FX_SEPIA_SETTINGS             ;\
             0-7   Sepia constants for Cr (00h..FFh)              ; Sepia
             8-15  Sepia constants for Cb (00h..FFh)              ;/
2765h   1  MODE_COMMON_MODE_SETTINGS_FILTER_MODE                   ;\
             Shadow register for 326Eh in SOC1                    ;
             0-2   UV Filter mode (0..7)                          ; Filter
             3-4   Y Filter mode  (0..3)                          ;
             5     Enable_y_filter (enable y permanently)         ;
             6     Threshold_switch, switch for adaptive Y filter threshold
             7     Off_switch, B/W filter enable switch           ;/
2766h   1  MODE_COMMON_MODE_SETTINGS_TEST_MODE   Test (00h..FFh)
             0-?  Test Pattern (0=None?, 1=Flat, 2=Ramp, 3=ColorBars,
                  4=VertStripes, 5=Noise, 6=HoriStripes)
             Output test pattern (instead camera image)
             requires "Refresh Command" sent to Sequencer
2767h  ..  RESERVED_MODE_67-68                  Reserved
```

DSi Aptina Camera Variables: HG (Histogram) (MCU:2Bxxh)

Histogram Variables HG (MCU:2B00h..2B61h)

```
2B00h  ..  RESERVED_HG_00-03        Reserved
2B04h   1  HG_MAX_DLEVEL            DarkLevel Limit (0..FFh, def=40h)
2B05h  ..  RESERVED_HG_05           Reserved
```

```
2B06h   1  HG_PERCENT               Percent?        (0..FFh, def=03h)
2B07h  ..  RESERVED_HG_07           Reserved
2B08h   1  HG_DLEVEL                DarkLevel       (0..FFh, def=10h)
2B09h  ..  RESERVED_HG_09-16        Reserved
2B17h   1  HG_AVERAGELUMA           Average Luma      (0..FFh)
2B18h  ..  RESERVED_HG_18-1A        Reserved
2B1Bh   2  HG_BRIGHTNESSMETRIC      Brightness Metric (0..FFFFh)
2B1Dh  ..  RESERVED_HG_1D           Reserved
2B1Fh   1  HG_LLMODE                Low Light mode controls (def=C4h)
             0-3  Brightness Metric Prescaler (01h..0Fh)
             4-5  Unused (0)
             6    HG_2d_corr_vs_clusterdc
             7    Clusterdc_vs_gains
2B20h   1  HG_LL_SAT1               LL_SAT1           (0..FFh, def=43h)
2B21h   1  UNDOC_HG_21              Whatever          (0..FFh, def=10h)
2B22h   1  HG_LL_APCORR1            LL_APCORR1        (0..FFh, def=03h)
2B23h   1  UNDOC_HG_23              Whatever          (0..FFh, def=04h)
2B24h   1  HG_LL_SAT2               LL_SAT2           (0..FFh, def=0Ch)
2B25h   1  HG_LL_INTERPTHRESH2      LL_INTERPTHRESH2  (0..FFh, def=23h)
2B26h   1  HG_LL_APCORR2            LL_APCORR2        (0..FFh)
2B27h   1  HG_LL_APTHRESH2          LL_APTHRESH2      (0..FFh, def=04h)
2B28h   2  HG_LL_BRIGHTNESSSTART    LL_BRIGHTNESSSTART (0..FFFFh, def=0A8Ch)
2B2Ah   2  HG_LL_BRIGHTNESSSTOP     LL_BRIGHTNESSSTOP  (0..FFFFh, def=34BCh)
2B2Ch   1  HG_NR_START_R            NR_START_R        (0..FFh, def=06h)
2B2Dh   1  HG_NR_START_G            NR_START_G        (0..FFh, def=0Eh)
2B2Eh   1  HG_NR_START_B            NR_START_B        (0..FFh, def=06h)
2B2Fh   1  HG_NR_START_OL           NR_START_OL       (0..FFh, def=06h)
2B30h   1  HG_NR_STOP_R             NR_STOP_R         (0..FFh, def=1Eh)
2B31h   1  HG_NR_STOP_G             NR_STOP_G         (0..FFh, def=1Eh)
2B32h   1  HG_NR_STOP_B             NR_STOP_B         (0..FFh, def=1Eh)
2B33h   1  HG_NR_STOP_OL            NR_STOP_OL        (0..FFh, def=1Eh)
2B34h   1  HG_NR_GAINSTART          NR_GAINSTART      (0..FFh, def=08h)
2B35h   1  HG_NR_GAINSTOP           NR_GAINSTOP       (0..FFh, def=80h)
2B36h   1  HG_CLUSTERDC_TH          CLUSTERDC_TH      (0..FFh, def=1Eh)
2B37h   1  HG_GAMMA_MORPH_CTRL      Gamma Morphing Control (0..FFh, def=3)
             0-1 Enable Gamma Morph (0=Disable, 1=Use Table A, 2=Use Table B,
                 3=AutoMorph between Table A and B based on BrightnessMetric)
             2-7 Unspecified
2B38h   2  HG_GAMMASTARTMORPH       Gamma Start Morph (0..FFFFh, def=0A8Ch)
2B3Ah   2  HG_GAMMASTOPMORPH        Gamma Stop Morph  (0..FFFFh, def=34BCh)
2B3Ch  19  HG_GAMMA_TABLE_A_0-18    Gamma Table A for normal light condition
            Default=xx,1B,2E,4C,78,98,B0,E8,CF,D9,E1,E8,EE,F2,F6,F9,FB,FD,FF
2B4Fh  19  HG_GAMMA_TABLE_B_0-18    Gamma Table B for low light condition
            Default=xx,0F,1A,2E,50,6A,80,91,A1,AF,BB,C6,D0,D9,E2,EA,F1,F9,FF
            Above 2 tables have normal byte-order (Entry0,Entry1,...,Entry18)
2B62h   2  HG_FTB_START_BM          (0..FFFFh, def=7FBCh) (REV3)
2B64h   2  HG_FTB_STOP_BM           (0..FFFFh, def=82DCh) (REV3)
2B66h   2  HG_CLUSTER_DC_BM         (0..FFFFh, def=4A38h) (REV3)
```

DSi Alternate Cameras from Unknown Manufacturer

Device A0h/E0h appear to be cameras from an alternate manufacturer. DSi games are supporting these devices, but as by now, there aren't any DSi consoles known to be actually fitted with these cameras.
The camera type & manufacturer are still unknown. Below initialization data is containing some characterstic info that should allow to identify them. For example, register 03h appears to be bank-switching the other registers.

unknown_cam_get_chip_id:
Reading an 8bit value from index 00h (in any bank?) seems to return some Chip ID, at least the DSi is reading that register before initialization (despite of reading it, the DSi does appear to ignore that value though).

Note: On a DSi with Aptina cameras, trying to read anything from IC2 devices A0h/E0h does just return FFh-bytes.

Formatting Note
Below tables consist of "Index,Length,Data[Length]" entries.

unknown_cam_type_code_list_init:

```
  003h, 1,001h              ;<-- bank maybe?
  009h, 3,0E2h,002h,002h
  004h, 1,010h
  004h, 1,0A0h
  004h, 2,090h,04Ch
  00Dh, 1,0FFh
  016h, 1,053h
  018h, 3,002h,001h,00Fh
  020h, 1,000h
  023h, 2,000h,000h
  034h, 8,000h,003h,000h,003h,001h,002h,000h,0C2h
  03Dh, 4,050h,050h,000h,067h
  042h, 1,01Ch
  04Ah, 2,043h,0F8h
  04Eh, 7,028h,0FCh,000h,024h,014h,008h,008h
  056h,13,000h,018h,028h,034h,044h,056h,06Eh,080h,0A4h,0C2h,0D6h,0E8h,0F4h
  065h,12,00Fh,038h,008h,000h,01Fh,01Fh,01Fh,01Fh,01Fh,01Fh,01Fh
  07Ah,17,039h,03Bh,03Ah,036h,03Ch,03Ch,03Ah,03Ch,03Ch,03Ch,03Ah,03Ch,038h
          03Ah,031h,03Ah,082h
  08Dh,22,08Ah,090h,096h,09Ch,0A4h,0AAh,0B0h,0B6h,0BCh,0C4h,0CAh,0D0h,0D6h
          0DCh,0E4h,0EAh,0F0h,0F2h,0F4h,0F6h,0F8h,0FAh
  0A9h, 1,02Bh
  0ABh, 3,02Eh,000h,050h
  0AFh, 1,070h
  0B2h, 4,03Ch,068h,049h,070h
  0B7h,21,032h,000h,00Eh,0F8h,00Ch,07Ah,040h,000h,000h,010h,044h,064h,052h
          012h,001h,0D7h,004h,002h,024h,002h,024h
  0D4h, 5,004h,004h,008h,00Ah,010h
  016h, 1,0F7h
  0DEh, 2,002h,024h
  016h, 1,053h
  0E1h, 1,034h
  0FFh, 1,00Fh
  003h, 1,002h              ;<-- bank maybe?
  005h, 2,06Dh,004h
  011h, 4,004h,048h,004h,048h
  016h, 2,00Ch,0D8h
  019h, 2,00Ch,0D8h
  01Eh, 6,002h,024h,070h,000h,001h,06Eh
  026h, 7,008h,00Fh,00Fh,006h,0FFh,0FFh,003h
  02Eh,19,07Eh,088h,074h,07Eh,008h,010h,080h,008h,084h,078h,001h,003h,00Ah
          025h,060h,0B0h,006h,000h,000h
  042h, 7,080h,010h,010h,010h,040h,080h,0FFh
  04Ah,30,000h,000h,001h,0E5h,001h,0E0h,000h,070h,002h,0F0h,000h,02Eh,001h
          0F3h,000h,005h,000h,000h,001h,000h,000h,0C0h,000h,026h,000h,01Ch
          000h,0B3h,000h,086h
  069h,36,000h,000h,006h,014h,014h,01Fh,000h,000h,000h,000h,000h,01Fh,000h
          000h,010h,010h,010h,01Fh,000h,000h,004h,004h,004h,01Fh,000h,000h
          000h,000h,000h,01Fh,000h,000h,010h,010h,010h,01Fh
  095h, 1,084h
  097h,18,002h,000h,0FFh,0FFh,000h,0FFh,0FFh,000h,000h,0FFh,0FFh,000h,0FFh
          0FFh,000h,0F8h,014h,010h
  0AAh,13,044h,098h,08Ch,09Ch,048h,08Ch,08Ah,09Ch,046h,02Ah,080h,008h,026h
  0B8h, 8,02Ah,084h,000h,026h,02Ah,080h,008h,020h
  0C1h,10,038h,020h,01Fh,01Dh,034h,020h,01Fh,01Dh,045h,05Dh
  0CCh, 2,020h,020h
```

```
  0D0h, 3,080h,000h,0FFh
  003h, 1,000h               ;<-- bank maybe?
  013h, 2,000h,04Ch
  01Dh, 2,000h,04Ch
  015h, 2,001h,05Fh
  055h, 2,001h,05Eh
  031h, 6,006h,068h,00Ch,005h,004h,047h
  047h, 2,000h,003h
  04Ah, 3,0A0h,000h,003h
  04Fh, 2,000h,003h
  059h, 2,000h,001h
  05Fh, 2,000h,001h
  066h, 1,09Eh
  06Eh, 2,07Fh,003h
  075h, 1,050h
  07Ah, 2,000h,001h
  07Eh, 1,020h
  082h, 1,038h
  084h,14,003h,040h,003h,040h,000h,000h,040h,003h,0FFh,002h,008h,020h,018h,006h
  093h,11,020h,040h,040h,01Fh,002h,000h,000h,000h,000h,000h,000h
  003h, 1,001h               ;<-- bank maybe?
  00Fh, 1,0C9h                    ;or, for Device E0h: 00Fh, 1,0C8h
  052h, 3,004h,008h,008h          ;or, for Device E0h: N/A
  003h, 1,002h               ;<-- bank maybe?
  026h, 1,008h                    ;or, for Device E0h: 026h, 1,000h
  0CCh, 2,0C0h,0C0h               ;or, for Device E0h: N/A
  0B4h, 1,000h                    ;or, for Device E0h: N/A
  0B6h, 1,026h                    ;or, for Device E0h: N/A
  0B9h, 3,000h,008h,026h          ;or, for Device E0h: N/A
  0BDh, 1,000h                    ;or, for Device E0h: N/A
  026h, 1,008h                    ;or, for Device E0h: N/A
  003h, 1,001h               ;<-- bank maybe?
  02Dh, 1,0FFh
  004h, 1,020h
```

unknown_cam_type_code_list_activate:

```
  003h, 1,002h               ;<-- bank maybe?
  0A7h, 1,014h
  003h, 1,001h               ;<-- bank maybe?
  004h, 1,0A0h
  004h, 1,090h
  02Dh, 1,000h
  004h, 1,098h
```

Random Note

This info is probably not really helpful, but the DSi firmware contains code for setting Register C1h..C9h (within unknown bank) to one of the following twelve settings.

```
  C1h, 8,038h,030h,01Fh,01Fh,02Ch,030h,01Fh,01Fh
  C1h, 8,038h,030h,01Fh,01Fh,038h,030h,01Fh,01Fh
  C1h, 8,02Ch,030h,01Fh,01Fh,02Ch,030h,01Fh,01Fh
  C1h, 8,02Ch,030h,01Fh,01Fh,02Ch,030h,01Fh,01Fh
  C1h, 8,02Ch,030h,01Fh,01Fh,02Ch,030h,01Fh,01Fh
  C1h, 8,02Ch,030h,01Fh,01Fh,02Ch,030h,01Fh,01Fh
  C1h, 8,030h,028h,018h,018h,034h,028h,008h,018h
  C1h, 8,030h,028h,018h,018h,030h,028h,008h,018h
  C1h, 8,028h,028h,018h,018h,028h,028h,008h,018h
  C1h, 8,028h,028h,018h,018h,028h,028h,008h,018h
  C1h, 8,028h,028h,018h,018h,028h,028h,008h,018h
  C1h, 8,028h,028h,018h,018h,028h,028h,008h,018h
```

DSi Cameras

Camera registers
Cameras are controlled and initialized via I2C bus (on ARM7 side).
DSi I2C Bus
The actual camera data transfers are done with below registers (on ARM9 side).

4004200h - DSi9 - CAM_MCNT - Camera Module Control

```
  0     Unknown                                                  (R or R/W)
  1     Unknown (1=Enable?)                                      (R or R/W)
  2-4   Unknown                                                  (R or R/W)
  5     Unknown (1=Enable?) (0=CamI2C fails?)                    (R or R/W)
  6     Unknown                                                  (R or R/W)
  7     Unknown (gets set automatically?)                              (R?)
  8-15  Unknown/Unused (00h)                                           (0?)
```

Written values are 0000h and 0022h.
Camera I2C access works only when bit5=1 (otherwise camera i2c reads just return FFh; maybe bit5=0 issues a reset to the camera devices or so?)
"Used for resetting cameras. Once cameras are reset by poking this register, all three 0x0400420X camera registers are set to zero." Uh, is that really true, and which "three" registers are that?

4004202h - DSi9 - CAM_CNT - Camera Control

```
  0-3   Number of DMA scanlines minus 1 (usually 3=Four Scanlines)   (R or R/W)
  4     Data request? or Data overrun? or so?                               (R)
  5     Clear bit4, and flush CAM_DAT till next Camera Vblank?              (W)
  6-7   Unknown/Unused (0)                                                 (0?)
  8-9    ? Set to 2 during init, 0 on cameras shutdown                    (R/W)
  10     ? Set to 1 during init, 0 on cameras shutdown                    (R/W)
  11    IRQ Enable      (0=Disable, 1=Enable)                             (R/W)
  12    Unknown/Unused (0)                                                 (0?)
  13    Color Format    (0=Direct/YUV422, 1=Convert YUV-to-RGB555)   (R or R/W)
  14    Trimming Enable (0=Normal/FullPicture, 1=Crop via SOFS/EOFS) (R or R/W)
  15    Transfer Enable (0=Disable/AllowConfig, 1=Enable/Transfer)        (R/W)
```

4004204h - DSi9 - CAM_DAT - Camera Data (R)
Transfers two camera pixels at once (from left-to-right, starting with upper scanline).
Pixel Format (in "YUV422" mode):

```
  0-7   First Pixel Luminance (Y)   (unsigned, 00h..FFh, FFh=white)
  8-15  Both Pixels Blue (Cb aka U) (unsigned, 00h..FFh, 80h=gray)
  16-23 Second Pixel Luminance (Y)  (unsigned, 00h..FFh, FFh=white)
  24-31 Both Pixels Red (Cr aka V)  (unsigned, 00h..FFh, 80h=gray)
```

Pixel Format (in YUV-to-RGB555 mode) (matches 2D Engine Bitmap format):

```
  0-4   First Pixel Red Intensity    (0..31)
  5-9   First Pixel Green Intensity  (0..31)
  10-14 First Pixel Blue Intensity   (0..31)
  15    First Pixel Alpha (always 1=NonTransparent)
  16-20 Second Pixel Red Intensity   (0..31)
  21-25 Second Pixel Green Intensity (0..31)
  26-30 Second Pixel Blue Intensity  (0..31)
  31    Second Pixel Alpha (always 1=NonTransparent)
```

The Aptina camera's MODE_OUTPUT_FORMAT registers and MIPI_CONTROL register can be configured to output stuff like YUV, RGB555, RGB444, BGR565, RAW8, etc. However, DSi games seem to be always using YUV mode at camera side (and the above RGB555 data is produced by activating YUV-to-RGB conversion in CAM_CNT.bit13 at console side).
YUV mode gives better quality with 8bit resolution (whilst RGB555 mode is having only 5bit, and, as it's converted from YUV, it's certainly having color information being shared for each two-pixel groups, too).
CAM_DAT should be usually read via NDMA (see below). Manually reading CAM_DAT for one block (eg. 256x4 pixels) does work, but it's unknown how to retrieve further blocks via manual reading (except, one further block arrives after around 40000h clock cycles, but that's much too slow, and it's only one extra block).

Formulas for converting YUV to RGB

```
R = Y+(Cr-80h)*1.402
G = Y-(Cr-80h)*0.714)-(Cb-80h)*0.344
B = Y+(Cb-80h)*1.772
```

Clip results to MinMax(00h,FFh), and apply final divide by 8 for RGB555.

4004210h - DSi9 - CAM_SOFS Camera Trimming Starting Position Setting (32bit)
4004214h - DSi9 - CAM_EOFS Camera Trimming Ending Position Setting (32bit)

```
0      Unused (0)                                                    (0)
1-9    X-Offset  (0..1FFh)   in words (ie. 2-pixel units)?        (R or R/W)
10-15  Unused (0)                                                    (0)
16-24  Y-Offset  (0..1FFh)   in scanlines?                        (R or R/W)
25-31  Unused (0)                                                    (0)
```

Crops the incoming camera picture before passing it to CAM_DAT, used only if enabled in CAM_CNT.14.

Write-Protected Camera Bits (R or R/W)

The "(R or R/W)" bits are getting Read-Only when camera transmission is enabled, ie. they can be changed only when CAM_CNT.Bit15=0.

Internal Camera Reflections from LCD Backlights

The LCD backlights can cause nasty reflections on the internal camera (particulary when wearing glasses). There isn't much that could be done during preview, but when taking photos, it might be recommended to output a black/dark picture on the LCDs during the capture.

Internal Camera Mirroring

The Internal Camera is conventionally having x-flip enabled (in Aptina MODE_SENSOR_READ_MODE registers), so the internal camera will behave as a mirror (which may appear more familar to most users in preview mode). The firmware's "Nintendo DSi Camera" utility is even saving jpg's in mirrored form instead of saving true authentic photos.

Camera Detection

There are four possible I2C camera devices, although usually only two of them should be installed. The firmware detects the cameras by reading their Chip ID registers (but without actually insisting on any specific ID values, instead, it's merely checking the ACK error flag in the I2C register - if all four devices are returning ACK=okay, then it's actually initializing all four cameras; though unknown if the GUI is actually supporting that many cameras).

Camera Init

```
[4004004h]=[4004004h] OR 0004h             ;SCFG_CLK, CamInterfaceClock = ON
[4004200h]=0000h, delay(1Eh)               ;CAM_MCNT, Camera Module Control
[4004004h]=[4004004h] OR 0100h, delay(1Eh) ;SCFG_CLK, CamExternal Clock = ON
[4004200h]=0022h, delay(2008h)             ;CAM_MCNT, Camera Module Control
[4004004h]=[4004004h] AND NOT 0100h        ;SCFG_CLK, CamExternal Clock = OFF
[4004202h]=[4004202h] AND NOT 8000h        ;CAM_CNT, allow changing params
[4004202h]=[4004202h] OR 0020h             ;CAM_CNT, flush data fifo
[4004202h]=([4004202h] AND NOT 0300h) OR 0200h
[4004202h]=[4004202h] OR 0400h
[4004202h]=[4004202h] OR 0800h             ;CAM_CNT, irq enable
[4004004h]=[4004004h] OR 0100h, delay(14h) ;SCFG_CLK, CamExternal Clock = ON
issue "aptina_code_list_init" via I2C bus on ARM7 side
[4004004h]=[4004004h] AND NOT 0100h        ;SCFG_CLK, CamExternal Clock = OFF
[4004004h]=[4004004h] OR 0100h, delay(14h) ;SCFG_CLK, CamExternal Clock = ON
issue "aptina_code_list_activate" via I2C bus on ARM7 side
[4004202h]=[4004202h] OR 2000h             ;CAM_CNT, enable YUV-to-RGB555
[4004202h]=([4004202h] AND NOT 000Fh) OR 0003h
[4004202h]=[4004202h] OR 0020h             ;CAM_CNT, flush data fifo
[4004202h]=[4004202h] OR 8000h             ;CAM_CNT, start transfer
[4004120h]=04004204h                       ;NDMA1SAD, source CAM_DTA
```

```
[4004124h]=0xxxxxxxh                    ;NDMA1DAD, dest RAM/VRAM
[4004128h]=00006000h                    ;NDMA1TCNT, len for 256x192 total
[400412Ch]=00000200h                    ;NDMA1WCNT, len for 256x4 blocks
[4004130h]=00000002h                    ;NDMA1BCNT, timing interval or so
[4004138h]=8B044000h                    ;NDMA1CNT, start camera DMA
```

Specifications
The Nintendo DSi contains two cameras. The cameras can be used in the Nintendo DSi Camera application or DSi games that are compatible.

```
640*480 VGA (0.3 Megapixel)
No zoom and no flash.
```

Photos saved in JPG format (saved in DCIM/ folder on the SD/SDHC or in the internal memory).

Camera Applications

```
Nintendo DSi Camera
System Menu (can take photos, and can display JPG's with "Star" sticker)
Flipnote (doesn't directly support camera hardware, but can import JPG's)
```

Camera Games

```
Asphalt 4 : Elite Racing (DSiWare)
Brain Challenge (DSiWare)
Classic Word Games
Cooking Coach
Pop SuperStar : Road To Celebrity (DSiWare)
Real Football 2009 (DSiWare)
WarioWare : Snapped! (DSiWare)
iCarly
Pokemon Black,White (2010,JP)
Castle of Magic (DSiWare)
Photo Dojo (DSiWare)
System Flaw (mis-uses camera as gyro sensor)
```

DSi SD/MMC Protocol and I/O Ports

I/O Ports
DSi SD/MMC I/O Ports: Command/Param/Response/Data
DSi SD/MMC I/O Ports: Interrupt/Status
DSi SD/MMC I/O Ports: Control Registers
DSi SD/MMC I/O Ports: Unknown/Unused Registers
DSi SD/MMC I/O Ports: Misc

SD/MMC Protocol
DSi SD/MMC Protocol: Command/Response/Register Summary
DSi SD/MMC Protocol: General Commands
DSi SD/MMC Protocol: Block Read/Write Commands
DSi SD/MMC Protocol: Special Extra Commands
DSi SD/MMC Protocol: CSR Register (32bit Card Status Register)
DSi SD/MMC Protocol: SSR Register (512bit SD Status Register)
DSi SD/MMC Protocol: OCR Register (32bit Operation Conditions Register)
DSi SD/MMC Protocol: CID Register (128bit Card Identification)
DSi SD/MMC Protocol: CSD Register (128bit Card-Specific Data)
DSi SD/MMC Protocol: EXT_CSD Register (4096bit Extended CSD Register) (MMC)
DSi SD/MMC Protocol: RCA Register (16bit Relative Card Address)
DSi SD/MMC Protocol: DSR Register (16bit Driver Stage Register) (Optional)
DSi SD/MMC Protocol: SCR Register (64bit SD Card Configuration Register)
DSi SD/MMC Protocol: PWD Register (128bit Password plus 8bit Password len)
DSi SD/MMC Protocol: State

DSi SD/MMC Protocol: Signals

SDIO Protocol
DSi SDIO Special SDIO Commands
DSi SDIO Memory and I/O Maps
DSi SDIO Common Control Registers (CCCR)
DSi SDIO Function Basic Registers (FBR)
DSi SDIO Card Information Structures (CIS)
The DSi is using SDIO for the new DSi Wifi interface,
DSi Atheros Wifi SDIO Interface
DSi Atheros Wifi Internal Hardware

Pinouts
AUX DSi SD/MMC Pin-Outs

DSi SD/MMC I/O Ports: Command/Param/Response/Data

4004800h/4004A00h - SD_CMD - Command and Response/Data Type (R/W)

```
  15    undoc   Unknown/undoc  (read/write-able)
  14    undoc   Security Cmd?  (0=Normal, 1=Whatever/Security?) (sdio?)
  13    undoc   Data Length    (0=Single Block, 1=Multiple Blocks)
  12    undoc   Data Direction (0=Write, 1=Read)
  11    NTDT    Data Transfer  (0=No data, 1=With data)
  10-8  REP2-0  Response Type  (0=Auto, 1..2=Unknown/Reserved, 3=None, 4=48bit,
                                5=48bit+Busy, 6=136bit, 7=48bitOcrWithoutCRC7)
  7-6   CMD1-0  Command Type   (0=CMD, 1=ACMD, 2..3=unknown, maybe GEN WR/RD?)
  5-0   CIX     Command Index  (0..3Fh, command index)
```

Setting Command Type to "ACMD" is (NOT!!!) automatically sending an APP_CMD prefix prior to the command number (but: unknown what RCA value it's sending in the APP_CMD's parameter field).
For Multiple Blocks, the hardware supports automatically sending STOP_TRANSMISSION after the last block.
DSi software is usually setting Response Type to "Auto", which is causing the hardware to use the correct response/data type for standard SD/MMC commands (bit11-13 are ignored/should be zero when using "Auto"; and maybe same for bit14-15?).
One exception is that the DSi firmware isn't using "Auto" for SDIO commands (maybe the hardware isn't aware of them; or it's unable to distinguish between read/write direction of CMD53, which would require examining the command's PARAM bits).
There are some differences between some SD and MMC commands, unknown if/how "Auto" is working in that cases; unknown if there's a SD-or-MMC mode select bit for that purpose in some configuration register (note: The DSi firmware uses manual config instead of Auto for CMD8, which differs for SD vs MMC).
Invalid values can cause ILA error (particulary on setting NTDT for CMD12, or for CMD's Response=None).
ILA error will also occur if an old CMD is still busy.

4004804h/4004A04h - SD_CMD_PARAM0-1 - Argument (32bit, 2 halfwords) (R/W)

```
  31-0  Parameter value for CMD
```

The parameter value should be written <before> sending the command via SD_CMD/SDIO_CMD.

400480Ch/4004A0Ch - SD_RESPONSE0-7 - Response (128bit, 8 halfwords) (R)
After sending a command, wait for the CMDRESPEND bit (IRQ_STATUS.bit0) to get set, then read the RESPONSE (if the command does have any response).
For normal 32bit responses:

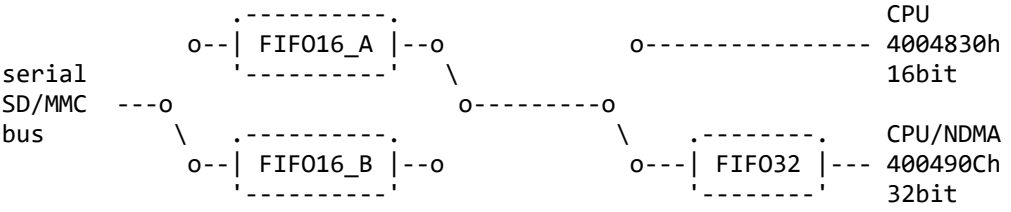
```
  31-0      Response
  127-32    Older Responses
```

For CID/CSD responses:

```
  119-0     120bit Response
  127-120   Zero (always?)
```

The above stuff is left-shifted when receiving new response bits (hence moving older responses to MSBs).

DATA16 vs DATA32
Data can be transferred in 16bit or 32bit units (as selected in DATA_CTL.bit1 and DATA32_IRQ.bit1). There are separate data, block len, and block count registers for 16bit and 32bit mode. 16bit mode uses two FIFOs (each with 200h-byte capactity). 32bit adds another FIFO (also with 200h-byte capacity):



The 32bit mode is some odd patchwork, apparently Nintendo/Toshiba considered it easier to add an extra 32bit FIFO (rather than to figure out how to add native 32bit access to Toshiba's original 16bit chip design).
The DSi firmware does use both 32bit and 16bit mode once and then; 32bit mode can be faster, and it's required for NDMA transfers (which don't support 16bit).

40048D8h/4004AD8h - SD_DATA_CTL

15-13	Always zero	
12	Unknown (usually 1)	(R?)
11-6	Always zero	
5	Unknown (read/write-able) (usually 0)	(R/W)
4	Unknown (usually 1)	(R?)
3-2	Always zero	
1	Select 16bit/32bit Data Mode (0=DATA16, 1=DATA32, see 4004900h)	(R/W)
0	Always zero	

DATA32 mode requires setting both 40048D8h.bit1 and 4004900h.bit1. For DATA16 mode, both bits should be zero (though DATA16 seems to be also working the same way when only either of the bits is zero).

400480Ah/4004A0Ah - SD_DATA16_BLK_COUNT - NumBlocks for 16/32 bit Modes (R/W)
4004908h/4004B08h - SD_DATA32_BLK_COUNT - NumBlocks for 32 bit Mode (R/W)

15-0	Number of Data Blocks for multiple read/write commands (0..FFFFh)

SD_DATA16_BLK_COUNT needs to be initialized in both 16bit and 32bit mode, the written value is copied to a internal register, which gets decremented after each block, and (when enabled in STOP_INTERNAL_ACTION.bit8) the hardware will automatically send STOP_TRANSMISSION (CMD12) after the last block (otherwise the hardware would keep transferring blocks infinitely).
For Data32 mode, DATA32_BLK_COUNT should be set to the same value (it doesn't really affect the transfer though, the register is intended only for watching the transfer progress: DATA32_BLK_COUNT is decremented after each block (when FIFO32 gets empty); except when it would become zero, in that case it stays stuck at 0001h).

4004826h/4004A26h - SD_DATA16_BLK_LEN - FIFO16 Size for 16/32 bit Modes (R/W)
4004904h/4004B04h - SD_DATA32_BLK_LEN - FIFO32 Size for 32 bit Mode (R/W)

15-10	Always zero
9-0	Data Block Length in bytes (for DATA16: clipped to max 0200h by hw)

SD_DATA16_BLK_LEN needs to be initialized in both 16bit and 32bit mode. For 32bit mode, SD_DATA32_BLK_LEN should be also set to the same value (otherwise odd effects might occur when forwarding FIFO16 to/from FIFO32).
The block length should be usually 0200h (for 512-byte SD/MMC memory blocks). Other values may be needed for SDIO functions, or when accessing SSR/SCR/PWD registers via data transfers.
DATA32_BLK_LEN can be set to max=3FFh (unlike DATA16_BLK_LEN which is clipped to max=200h by hardware), though settings bigger than 200h won't work in practice (since the FIFOs are only 200h bytes in size).

4004830h/4004A30h - SD_DATA16_FIFO - Data FIFO for 16bit Mode (R/W)

400490Ch/4004B0Ch - SD_DATA32_FIFO - Data FIFO for 32bit Mode (R/W)
For Data16:
```
  15-0   Data (Read/Write one block (usually 100h halfwords) upon RXRDY/TXRQ)
```
For Data32:
```
  31-0   Data (Read/Write one block (usually 80h words) upon RX32RDY/TX32RQ)
```
See the RXRDY/TXRQ and RX32RDY/TX32RQ interrupt flags for details.
The first Data32 block may be written even before sending the command (the DSi firmware is actually doing that, although, performance-wise, it would make sense only when writing multiple clusters, ie. sending the first block of the next cluster while the previous write is still in progress).
Observe that RX32RDY/TX32RQ are actually FIFO full/empty flags (getting triggered any time when FIFO is full/empty, so you must know for yourself if you want to transfer data in that situations; eg. FIFO empty will trigger even after having written all data blocks, or when not intending to write any data at all).

DSi SD/MMC I/O Ports: Interrupt/Status

All SD/MMC IRQs are triggering IF2.8, CardIRQ does ADDITIONALLY trigger IF2.9
All SDIO IRQs are triggering IF2.10, CardIRQ does ADDITIONALLY trigger IF2.11?

400481Ch/4004A1Ch - SD_IRQ_STATUS0-1 - Interrupt Status (R/ack)
4004820h/4004A20h - SD_IRQ_MASK0-1 - Interrupt Mask (R/W)
The IRQ_STATUS registers contain acknowledge-able IRQ Flags (those bits that that are maskable in IRQ_MASK register), as well as static read-only status bits without IRQ function (eg. WRPROTECT).
IRQ Flags/Write (0=Acknowledge, 1=No change)
IRQ Flags/Read (0=No IRQ, 1=IRQ)
IRQ Mask (0=Enable, 1=Disable) (8B7F031Dh when all IRQs disabled)
```
  Bit Stat  Mask  Function
  0   SREP  MREP  CMDRESPEND    (response end) (or R1b: busy end)
  1   0     0     Unknown/unused (always 0)
  2   SRWA  MRWA  DATAEND       (set after (last) data block end)
  3   SCOT  MCOT  CARD_REMOVE (0=No event, 1=Is/was newly ejected)      ;\
  4   SCIN  MCIN  CARD_INSERT (0=No event, 1=Is/was newly inserted)     ; SD
  5   undoc 0     SIGSTATE    (0=Ejected, 1=Inserted) (SDIO: always 1)  ; Slot
  6   0     0     Unknown/unused (always 0)                            ; Sw's
  7   undoc 0     WRPROTECT   (0=Locked/Ejected, 1=Unlocked/HalfEjected);/
  8   undoc undoc CARD_REMOVE_A (0=No event, 1=High-to-Low occurred)    ;\SD
  9   undoc undoc CARD_INSERT_A (0=No event, 1=Low-to-High occurred)    ; Slot
  10  undoc 0     SIGSTATE_A    (usually 1=High) ;also as so for SDIO   ;/Data3
  11  0     0     Unknown/unused (always 0)
  12  0     0     Unknown/unused (always 0)
  13  0     0     Unknown/unused (always 0)
  14  0     0     Unknown/unused (always 0)
  15  0     0     Unknown/unused (always 0)
  16  SCIX  MCIX  CMD_IDX_ERR   Bad CMD-index in response      (RCMDE,SCMDE)
  17  SCRC  MCRC  CRCFAIL       CRC response error (WCRCE,RCRCE,SCRCE,CCRCE)
  18  SEND  MEND  STOPBIT_ERR   End bit error      (WEBER,REBER,SEBER,CEBER)
  19  SDTO  MDTO  DATATIMEOUT   Data Timeout               (NRCS,NWCS,KBSY)
  20  SFOF  MFOF  RXOVERFLOW    HOST tried write full
  21  SFUF  MFUF  TXUNDERRUN    HOST tried read empty
  22  SCTO  MCTO  CMDTIMEOUT    Response start-bit timeout         (NRS,NSR)
  23  ???   0     Unknown/undoc (usually set) (zero after sending TX data?)
  24  SBRE  MBRE  RXRDY         (fifo not empty) (request data read)
  25  SBWE  MBWE  TXRQ          (datafifoempty?) (request data write)
  26  0     0     Unknown/unused (always 0)
  27  undoc undoc Unknown/undoc  (bit27 is mask-able in IRQ_MASK)
  28  0     0     Unknown/unused (always 0)
  29  undoc 0     CMD_READY? (inverse of BUSY?) (unlike toshiba ILFSL/IFSMSK)
  30  undoc 0     CMD_BUSY   (CMD_BUSY=0 shortly before CMD_READY=1?)
  31  ILA   IMSK  Illegal Command Access (old CMD still busy, or wrong NTDT)
```

Normally, IRQs should be acknowledged by writing "FLAGS=NOT X", whilst the firmware is using an unstable "FLAGS=FLAGS AND NOT X" read-modify-write function (accidentally acknowledging any IRQs that have newly occurred during that operation).

4004836h/4004A36h - SD_CARD_IRQ_STAT (R/ack)
4004838h/4004A38h - SD_CARD_IRQ_MASK (R/W)
IRQ Flags/Write (0=Acknowledge, 1=No change)
IRQ Flags/Read (0=No IRQ, 1=IRQ)
IRQ Mask (0=Enable, 1=Disable) (C007h when all IRQs disabled)

```
Bit  Stat  Mask  Function
15   undoc undoc  SomeIRQ (triggered SOMETIMES on forced CMDTIMEOUT?)
14   undoc undoc  SomeIRQ (triggered near DATAEND?)
13-3 0     0      Always zero
2    undoc undoc  SomeIRQ (triggered on forced TXUNDERRUN?)
1    undoc undoc  SomeIRQ (triggered about once per datablock?)
0    CINT0 CIMSK0 CardIRQ (triggered by /IRQ aka Data1 pin; for SDIO devices)
```

All stat bits (except bit1) are triggered only if enabled in SD_CARD_IRQ_CTL.
Bit0 is actually used (for SDIO hardware), the other bits aren't used by existing software (they don't seem to be useful; purpose might be error testing, or forcing commands to abort).

4004834h/4004A34h - SD_CARD_IRQ_CTL (R/W)
CardIRQ Enable works only when also writing SD_CARD_PORT_SELECT.bit10=0 and only with valid SD_CARD_CLK_CTL setting.

```
15-10  Always zero
9      Enable setting SD_CARD_IRQ_STAT.bit14 and cause nothing special? (R/W)
8      Enable setting SD_CARD_IRQ_STAT.bit15 and cause CMDTIMEOUT?      (R/W)
7-3    Always zero
2      Enable setting SD_CARD_IRQ_STAT.bit2 and cause TXUNDERRUN?       (R/W)
1      Always zero
0      Enable setting SD_CARD_IRQ_STAT.bit0 (CardIRQ upon Data1=LOW)    (R/W)
```

Bit9 is autocleared at time when bit9 causes setting SD_CARD_IRQ_STAT.bit14.
Bit2 is autocleared shortly before bit8 causes CMDTIMEOUT.
SD_CARD_IRQ_STAT.bit15 is set only when setting bit8 AND bit2 DURING cmd/xfer.

40048F8h/40048FAh - SD_EXT_IRQ_STAT0-1 - Interrupt Status (R/ack)
40048FCh/40048FEh - SD_EXT_IRQ_MASK0-1 - Interrupt Mask (R/W)
The 3DS bootroom is using bit0-2 for eMMC insert/eject detection. DSi software isn't doing such stuff (though the registers seem to exist in DSi hardware, too). Either way, eMMC cannot be ejected on neither DSi nor 3DS, so the feature is kinda useless.

```
Bit   Stat  Mask  Function
0     SCOT  MCOT  CARD_REMOVE (0=No event, 1=Is/was newly ejected)     ;\eMMC
1     SCIN  MCIN  CARD_INSERT (0=No event, 1=Is/was newly inserted)    ; Slot
2     undoc 1     SIGSTATE    (MMC: Always 1=Inserted) (SDIO: always 0);/Sw
3     0     RW    Unknown (stat always 0, but mask is R/W)   ;\maybe another
4     0     RW    Unknown (stat always 0, but mask is R/W)   ; unimplemented
5     0     1     Unknown (stat always 0, and mask always 1) ;/device?
6     0     RW    Unknown (stat always 0, but mask is R/W)   ;\maybe yet one
7     0     RW    Unknown (stat always 0, but mask is R/W)   ; more?
8-15  0     0     Unknown/unused (always 0)                  ;/
16    ??    RW    Unknown (stat can be 0/1)               (R/W? or rather R?)
17    ??    RW    Unknown (stat can be 0/1)               (R/W? or rather R?)
18    ??    1     Unknown (1=normal, 0=data/read from card to fifo busy?) (R)
19    0     RW    Unknown (stat always 0, but mask is R/W)
20    0     RW    Unknown (stat always 0, but mask is R/W)
21    0     1     Unknown (stat always 0, and mask always 1)
22    0     RW    Unknown (stat always 0, but mask is R/W)
23    0     RW    Unknown (stat always 0, but mask is R/W)
24-31 0     0     Unknown/unused (always 0)
```

Some of the unknown bits might be CD/Data3 or IRQ/Data1 pins for Device 1, in similar way as for Device 0 (maybe that is in bit16-18, which are observed to become nonzero in certain situations).
EXT_IRQ_STAT can be 00000000h..00070004h for MMC, or always 00000000h for SDIO (as there's only one device on SDIO bus). EXT_IRQ_MASK is R/W (can be 00240024h..00FF00FFh) for both SD/MMC and SDIO.

4004900h/4004B00h - SD_DATA32_IRQ

```
15-13  Always zero
12     TX32RQ IRQ Enable   (0=Disable, 1=Enable)                      (R/W)
11     RX32RDY IRQ Enable  (0=Disable, 1=Enable)                      (R/W)
10     Clear FIFO32        (0=No change, 1=Force FIFO32 Empty)        (W)
9      TX32RQ IRQ Flag     (0=IRQ, 1=No) (0=FIFO32 Empty)             (R)
8      RX32RDY IRQ Flag    (0=No, 1=IRQ) (1=FIFO32 Full)              (R)
7-2    Always zero
1      Select 16bit/32bit Data Mode (0=DATA16, 1=DATA32, see 40048D8h) (R/W)
0      Always zero
```

Bit8,9 are extra IRQ flags, the flags get set ONLY in DATA32 mode (not in DATA16 mode).
Bit8,9 are somewhat edge-triggered (setting the IF2 bit only on NoIRQ-to-IRQ transitions; whilst Disable-to-Enable transitions don't trigger IF2).
Bit8,9 don't need to be acknowledged, they are automatically switched to "No IRQ" by hardware (when reading/writing DATA32_FIFO, ie. when the FIFO is no longer empty/full).

400482Ch/4004A2Ch - SD_ERROR_DETAIL_STATUS0-1 - Error Detail Status (R)
This register contains extra info about the error bits in SD_IRQ_STATUS. The error bits (except bit13/always set) are automatically cleared when sending a new command by writing to SD_CMD.

```
31-23 0      Always zero
22    KBSY   Timeout for CRC status busy                          ;\STAT.19
21    NWCS   Timeout for CRC status   (can occur for Data Write)  ; (SDTO)
20    NRCS   Timeout for Data start-bit, or for Post Data Busy    ;/
19-18 0      Always zero
17    NRS    Response Timeout for auto-issued CMD12               ;\STAT.22
16    NCR    Response Timeout for non-auto-issued CMD's           ;/(SCTO)
15-14 0      Always zero
13    undoc  Unknown/undoc (always 1)                            ;-Always 1
12    0      Always zero
11    WCRCE  CRC error for Write CRC status for a write command   ;\
10    RCRCE  CRC error for Read Data                              ; STAT.17
9     SCRCE  CRC error for a Response for auto-issued CMD12       ; (SCRC)
8     CCRCE  CRC error for a Response for non-auto-issued CMD's   ;/
5     WEBER  End bit error for Write CRC status                   ;\
4     REBER  End bit error for Read Data                          ; STAT.18
3     SEBER  End bit error for Response for auto-issued CMD12     ; (SEND)
2     CEBER  End bit error for Response for non-auto-issued CMD's ;/
1?    SCMDE  Bad CMD-index in Response of auto-issued CMD12       ;\STAT.16
0     RCMDE  Bad CMD-index in Response of non-auto-issued CMD's   ;/(SCIX)
```

Note: CMD12 is STOP_TRANSMISSION (automatically sent after BLK_COUNT blocks).
The four "auto-issued CMD12" bits exist for SD registers only (not for SDIO, going by old toshiba datasheets; which may be wrong).
SCMDE is probably in bit1 (though, official specs say bit0, which would be same as RCMDE).
Some error bits can be intentionally provoked: Bit8=1 when programming the controller to expect GET_STATUS to return a 136bit response. Bit16=1 when sending GET_CID in "tran" state. Bit20=1 when sending GET_STATUS configured to expect a data/read reply. Bit21=1 when sending GET_STATUS configured to expect a data/write block (and with actually sending a data block to it).

40048F6h/4004AF6h - SD_WRPROTECT_2 (RESERVED4) (R)

```
15-1   Always zero
0      WRPROTECT_2 for onboard eMMC (usually/always 0=Unlocked)          (R)
```

Bit0 is write-protect flag for onboard eMMC (equivalent to the SD/MMC slot's write-protect switch in 400481Ch.bit7, but in inverted form: 0=Unlocked for eMMC, instead of 0=Locked for SD/MMC). The firmware does check bit0 (and, if set, hangs shortly before starting games), unknown if the TWL CPU and DSi mainboard

do actually have any solder pads for it.

IRQ Edge-Triggering
One nasty "feature" for both IRQ_STATUS and DATA32_IRQ is that the interrupts are edge-triggered (IF2.bit8 gets set only on No-IRQ-to-IRQ transitions) (IF2 can be acknowledged even if IRQ(s) are still requested, which would mean that those IRQ(s) would get lost). Workaround would be:

```
- first acknowledge IF2.bit8 (must be done before next step)
- then check for pending IRQs in IRQ_STATUS and DATA32_IRQ, and process
  all of them
```

Ie. if you would process only a single IRQ, then any other IRQs would get lost.
For IRQ_STATUS, one could also force unprocessed IRQs to re-trigger IF2 by temporarily disabling IRQ_MASK bits (disable-to-enable for pending IRQs is also edge-triggering IF2). That trick works for IRQ_STATUS only, not for DATA32_IRQ.

DSi SD/MMC I/O Ports: Control Registers

4004802h/4004A02h - SD_CARD_PORT_SELECT

```
  15-11 Always zero
  10    Unknown (should be set on write) (reads as zero) (1=CardIRQ off!) (W)
  9-8   Unknown (Always 2 for SD/4004802h, always 1 for SDIO/4004A02h)    (R)
  7-4   Always zero
  3-1   Unknown (read/write-able)                                         (R/W)
  0     Port Select (0=SD Card Slot, 1=Onboard eMMC) (for SDIO: Unknown)  (R/W)
```

Known written values are 0400h and 0401h for SD/MMC. The register is left uninitialized for SDIO.

4004828h/4004A28h - SD_CARD_OPTION - Card Option Setup

```
  15   undoc Bus Width (0=4bit, 1=1bit)                                   (R/W)
  14   undoc Unknown (usually set)                                        (R?)
  13-9 0     Always zero
  8    undoc Unknown (firmware tries to toggle this after CLK change?)    (W?)
  7-4  RTO   Data start/busy timeout (2000h SHL 0..14, or 15=100h SDCLK's)(R/W)
  0-3  TO?   Unknown (another timeout, maybe for SDIO? in 32KHz units?)   (R/W)
```

See Timeout Notes below for details.

4004824h/4004A24h - SD_CARD_CLK_CTL Card Clock Control

```
  15-11 Always zero          ;unlike Toshiba: no HCLK divider-disable in bit15)
  10    Unknown (0=Normal, 1=Unknown, doesn't affect SDCLK output?)       (R/W)
  9     SDCLK Freeze     (0=Normal, 1=Freezes SDCLK output)               (R/W)
  8     SDCLK Pin Enable (0=Force SDCLK=LOW, 1=Output SDCLK=HCLK/n)       (R/W)
  7-0   HCLK Div (0,1,2,4,8,16,32,64,128 = Div2,4,8,16,32,64,128,256,512) (R/W)
```

The DSi uses HCLK=33.513982 MHz, the SDCLK pin can range from HCLK/512=65kHz to HCLK/2=16.757MHz, max transfer rate would be thus 8MByte/s in 4bit mode.
Max CLK speed:
Observe that card detection/initialization should be done at lower CLK rate than during normal operation.
For SD/MMC initialization: The DSi firmware starts with HCLK/128=262kHz (max allowed would be 400kHz for MMC). This is actually required: The DSi's onboard Samsung KMAPF0000M-S998 eMMC chip won't respond to ALL_GET_CID when trying to use 16MHz CLK). Higher CLK can be used once when detecting max speed (see TRAN_SPEED in CSD register; when extracting bits from CSD: mind the different 120bit-without-CRC vs 128bit-with-CRC notations).
For SDIO/Wifi initialization: The DSi firmware starts with HCLK/256=131kHz, and switches to HCLK/2=16.757MHz after reading SDIO Bus Speed register (Function0:00013h).
After init, one can use the detected speeds (see above), it should be also safe to assume that HCLK/2=16.757MHz is always supported after initialization (all SD devices should support at least 25MHz, and all(?) MMC devices at least 26MHz, and all DSi SDIO/Wifi boards should be fast enough either).
Notes:
The SDCLK pins are permanently pulsed, even for devices deselected via SD_CARD_PORT_SELECT.0, and

even if no CMD or DATA is being transferred. However, the DSi firmware is usually stopping SDCLK via Bit8=0 when not accessing SD/MMC (doing so may reduce noise and power consumption).
Trying to set bit9, or to set more than one bit in bit7-0 will freeze the SDCLK output (in this case SDCLK may get stuck HIGH or LOW, unlike Bit8=0 which forces LOW).
Odd Effect: Setting bit10-8 to ALL ones, combined with an invalid HDIV (eg. writing 0703h) does disable CardIRQ on Data1 pin.

4004808h/4004A08h - SD_STOP_INTERNAL_ACTION

```
15-9   Always zero
8      Auto-Stop (1=Automatically send CMD12 after BLK_COUNT blocks)  (R/W)
7-1    Always zero
0      Unknown (firmware often clears this bit, but never sets it?)   (R/W)
```

Existing code does set bit8 (prior to changing SD_DATA16_BLK_COUNT).
Existing code does clear bit0 (alongsides with IRQ enable/acknowledge or so).
Note: Bit8 MUST be also set for SDIO with multiple blocks (although SDIO doesn't literally have CMD12).

40048E0h/4004AE0h - SD_SOFT_RESET - Software Reset

```
15-3 Always zero
2    Unknown (always 1)                                          (R?)
1    Unknown (always 1) (though firmware tries to toggle this bit) (R?)
0    SRST Soft Reset (0=Reset, 1=Release)                        (R/W)
```

Software should apply reset after sensing card insertion/removal, and (thereafter) release reset in case of card insertion. Software reset does acknowledge all IRQs (except that from SDIO /IRQ pin?), and does probably also reinitialize some other registers.
Clearing bit0 does force following settings (while and as long as Bit0=0):

```
SD_STOP_INTERNAL_ACTION   = 0000h
SD_RESPONSE0-7            = zerofilled
SD_IRQ_STATUS0-1          = all IRQ flags acknowledged
SD_ERROR_DETAIL_STATUS0-1 = all bits cleared (except bit13/always set)
SD_CARD_CLK_CTL           = bit 8 and 10 cleared
SD_CARD_OPTION            = 40EEh
SD_CARD_IRQ_STAT          = 0000h
Internal FIFO16 address is reset to first halfword of FIFO_A
Reading FIFO16 returns 0000h (but old content reappears when releasing reset)
```

All other registers seem to be left unaffected (including the extra IRQ flags in 4004900h); though there may be some further hidden effects (like aborting transfers or resetting internal registers).
Note: The DSi firmware does issue reset by toggling both bit0 and bit1, although bit1 does seem to be read-only (always 1), and trying to clear that bit doesn't seem to have any effect at all.

Timeout Notes

Timeouts are counted in SDCLK units (the CLK-Pin rate selected in SD_CARD_CLK_CTL register). For Response-Timeouts, the timeout is fixed: Around 290h SDCLK's (preceeded by 30h SDCLK's for sending the command). For Data-Timeouts, the timeout can be selected in SD_CARD_OPTION.bit4-7, which is apparently what toshiba tried to describe as "RTO" bits. Values 0..14 select timeout "2000h SHL 0..14 SDCLK's" and value 15 selects "100h SDCLK's" (that, oddly, resulting in Data-timeout getting triggered before Response-timeout, which is rather nonsense since it's opposite of the actual transfer order).
For data/read, the timeout starts counting after transferring Command+Response. For data/write it starts after transferring Command+Response+DataBlock. The maximum duration for data timeouts (with RTO=14) would be around 8 seconds (at SDCLK=HCLK/2), or up to about 30 minutes (at HCLK/512).
One odd effect is that Response-Timeouts can occur (after 290h SDCLKs, and recursing the selected SDCLK=HCLK/n rate) even if SDCLK is stopped via SD_CARD_CLK_CTL.Bit8 (ie. the selected clock is kept running internally, and only the CLK-Pin output is forced LOW when Bit8=0).

DSi SD/MMC I/O Ports: Unknown/Unused Registers

Below registers don't seem to be used by existing software...

40048F2h/4004AF2h - Can be 0003h

```
15-2   Always zero
1-0    Unknown (0..3)                                          (R/W)
```

40048F4h/4004AF4h - Can be 0770h

```
15-11  Always zero
10-8   Unknown (0..7)                                          (R/W)
7      Always zero
6-4    Unknown (0..7)                                          (R/W)
3-0    Always zero
```

Unused Registers with Fixed value (all bits read-only, or write-only)

```
400482Ah/4004A2Ah 2    Fixed always zero?
4004832h/4004A32h 2    Fixed always zero?   ;(TC6371AF:BUF1 Data MSBs?)
400483Ah/4004A3Ah 2    Fixed always zero?   ;(SDCTL_SDIO_HOST_INFORMATION)
400483Ch/4004A3Ch 2    Fixed always zero?   ;(SDCTL_ERROR_CONTROL)
400483Eh/4004A3Eh 2    Fixed always zero?   ;(TC6387XB: LED_CONTROL)
4004840h/4004A40h 2    Fixed always 003Fh?
4004842h/4004A42h 2    Fixed always 002Ah?
4004844h/4004A44h 6Eh  Fixed always zerofilled?
40048B2h/4004AB2h 2    Fixed always FFFFh?
40048B4h/4004AB4h 6    Fixed always zerofilled?
40048BAh/4004ABAh 2    Fixed always 0200h?
40048BCh/4004ABCh 1Ch  Fixed always zerofilled?
40048DAh/4004ADAh 6    Fixed always zerofilled?
40048E2h/4004AE2h 2    Fixed always 0009h?  ;(RESERVED2/9, TC6371AF:CORE_REV)
40048E4h/4004AE4h 2    Fixed always zero?
40048E6h/4004AE6h 2    Fixed always zero?   ;(RESERVED3, TC6371AF:BUF_ADR)
40048E8h/4004AE8h 2    Fixed always zero?   ;(TC6371AF:Resp_Header)
40048EAh/4004AEAh 6    Fixed always zerofilled?
40048F0h/4004AF0h 2    Fixed always zero?   ;(RESERVED10)
4004902h/4004B02h 2    Fixed always zero?
4004906h/4004B06h 2    Fixed always zero?
400490Ah/4004B0Ah 2    Fixed always zero?
4004910h/4004B10h F0h  Fixed always zerofilled?
```

DSi SD/MMC I/O Ports: Misc

Toshiba Chips

The DSi SDIO/MMC port addresses and status bits appear to be identical to those on Toshiba SD/MMC/SDIO controller chips.

One small difference is that the DSi can set SD_IRQ_MASK.Bit27 (which wasn't used on (older) Toshiba chips). The Toshiba chips seem to include additional "CNF" configuration registers (which seem to be missing on DSi).

```
Chip             Year   Pages Features
Toshiba TC6371AF 2000-2002 58 SD/MMC/Smart/PCI (old/basic specs, no SDIO)
Toshiba TC6380AF 2001-2002 90 SD/MMC/SDIO/SmartMedia
Toshiba TC6387XB 2001-2002 62 SD/MMC/SDIO/SDLED
Toshiba TC6391XB 2002     202 SD/MMC/SDIO/SmartMedia/USB/LCD/etc.
Toshiba TC6393XB ?  ;\unknown features, no datasheet exists (the chips
Toshiba T7L66XB  ?  ;/are mentioned in tmio_mmc.h and tmio_mmc.c source)
```

The TC6380AF/TC6387XB/TC6391XB datasheets are more or less identical on the SD/MMC/SDIO section, TC6387XB is probably the best reference because it doesn't contain offtopic extras like SmartMedia, USB, LCD, etc. The datasheets contain I/O Maps with port addresses, but no description tables for the bits in those ports (though some bits are mentioned here and there in the text, scattered across many different pages, and other bits are left completely undocumented).

DSi SD/MMC Protocol: Command/Response/Register Summary

Basic Commands (class 0)

```
CMD0     sd/mmc spi GO_IDLE_STATE (CMD0 with arg=stuff) (type=bc)
CMD0     mmc        GO_PRE_IDLE_STATE (CMD0 with arg=F0F0F0F0h) (type=bc)
CMD0     mmc        BOOT_INITIATION (CMD0 with arg=FFFFFFFAh, type=N/A)
CMD1     sd/mmc spi SEND_OP_COND (On SD Cards: SPI only)
CMD2     sd/mmc     ALL_GET_CID (type=bcr)
CMD3     sd         GET_RELATIVE_ADDR (type=bcr)
CMD3     mmc        SET_RELATIVE_ADDR (type=ac)
CMD4     sd/mmc     SET_DSR (type=bc)
CMD5     sd     spi Reserved for I/O cards (see "SDIO Card Specification")
CMD5     mmc    ?   SLEEP_AWAKE (type=ac) (MMC only, IO_SEND_OP_COND on SDIO)
CMD7     sd/mmc     SELECT_DESELECT_CARD (type=ac) ;actually: (type=bcr)
CMD8     sd     spi SET_IF_COND (type=bcr)
CMD8     mmc    spi GET_EXT_CSD (type=adtc)
CMD9     sd/mmc spi GET_CSD (type=ac) (SPI: type=adtc)
CMD10    sd/mmc spi GET_CID (type=ac) (SPI: type=adtc)
CMD11    sd         VOLTAGE_SWITCH (type=ac)
CMD12    sd/mmc spi STOP_TRANSMISSION (type=ac)
CMD13    sd/mmc spi GET_STATUS (type=ac) (sends 16bit status in SPI Mode)
CMD14    mmc        BUSTEST_R (type=adtc) (MMC only, Reserved on SD)
CMD19    mmc        BUSTEST_W (type=adtc) (MMC only, SET_TUNING_BLOCK on SD)
CMD15    sd/mmc     GO_INACTIVE_STATE (type=ac)
```

Block-Oriented Read Commands (class 2)

```
CMD16    sd/mmc spi SET_BLOCKLEN (type=ac)
CMD17    sd/mmc spi READ_SINGLE_BLOCK (type=adtc)
CMD18    sd/mmc spi READ_MULTIPLE_BLOCK (type=adtc)
CMD19    sd         SET_TUNING_BLOCK (type=adtc)
CMD20    sd         SPEED_CLASS_CONTROL (type=ac)
CMD22    sd         Reserved
CMD23    sd/mmc-spi SET_BLOCK_COUNT (type=ac) (SPI supported ONLY on MMC?)
```

Block-Oriented Write Commands (class 4)

```
CMD16    sd/mmc spi SET_BLOCKLEN (type=ac)
CMD20    sd         SPEED_CLASS_CONTROL (type=ac)
CMD23    sd/mmc-spi SET_BLOCK_COUNT (type=ac) (SPI supported ONLY on MMC?)
CMD24    sd/mmc spi WRITE_BLOCK (type=adtc)
CMD25    sd/mmc spi WRITE_MULTIPLE_BLOCK (type=adtc)
CMD26    sd/mmc     Reserved For Manufacturer (MMC: PROGRAM_CID)
CMD27    sd/mmc spi PROGRAM_CSD (type=adtc)
```

Block-Oriented Write-Protection Commands (class 6)

```
CMD28    sd/mmc spi SET_WRITE_PROT (type=ac)
CMD29    sd/mmc spi CLR_WRITE_PROT (type=ac)
CMD30    sd/mmc spi GET_WRITE_PROT (type=adtc)
CMD31    -          SD: Reserved
CMD31    mmc        MMC: SEND_WRITE_PROT_TYPE (type=adtc)
```

Erase Commands (class 5)

```
CMD32    sd     spi ERASE_WR_BLK_START (type=ac)
CMD33    sd     spi ERASE_WR_BLK_END (type=ac)
CMD32-34 mmc    spi Reserved for compatibility with older MMC cards (uh?)
CMD35    mmc    spi ERASE_GROUP_START (type=ac)
CMD36    mmc    spi ERASE_GROUP_END (type=ac)
CMD37    mmc    spi Reserved for compatibility with older MMC cards (uh?)
CMD38    sd/mmc spi ERASE (type=ac)
CMD39    -          Reserved
CMD41    -          Reserved
```

Lock Card (class 7)

```
CMD16    sd/mmc spi SET_BLOCKLEN (type=ac)
CMD40    sd         Defined by DPS Spec (Data Protection System) (type=adtc)
CMD42    sd/mmc spi LOCK_UNLOCK (type=adtc)
CMD43-47 -          Reserved
CMD51    -          Reserved
```

Application-Specific Commands (class 8)

```
CMD39-40 mmc        MMCA Optional Command, currently not supported
CMD55-56 mmc        MMCA Optional Command, currently not supported
CMD55    sd     spi APP_CMD (type=ac)    ;\also defined for MMC,
CMD56    sd     spi GEN_CMD (type=adtc)  ;/but ONLY in SPI mode !!??
CMD60-63 sd/mmc spi Reserved for manufacturer
```

I/O Mode Commands (class 9) (Refer to "SDIO Card Specification")

```
CMD5     sdio   spi SDIO: IO_SEND_OP_COND
CMD52    sdio   spi SDIO: IO_RW_DIRECT
CMD53    sdio   spi SDIO: IO_RW_EXTENDED
CMD54    -           SDIO: Reserved
CMD39    mmcio       MMCIO: FAST_IO (type=ac)
CMD40    mmcio       MMCIO: GO_IRQ_STATE (type=bcr)
```

Switch Function Commands (class 10) (version 1.10+)

```
CMD6     mmc    spi SWITCH      (type=ac)   ;related to EXT_CSD register
CMD6     sd     spi SWITCH_FUNC (type=adtc)
CMD34-37 sd+spi     Reserved for Command Systems from CMD6  ;\SPI
CMD50,57 sd+spi     Reserved for Command Systems from CMD6  ;/
CMD34-35 sd         Reserved                                ;\
CMD36-37 sd         Undoc (description field is held blank) ; Non-SPI
CMD50,57 sd         Undoc (description field is held blank) ;/
```

Function Extension Commands (class 11)

```
CMD21    sd         Reserved for DPS Specification (Data Protection System)
CMD48    sd         READ_EXTR_SINGLE (type=adtc)
CMD49    sd         WRITE_EXTR_SINGLE (type=adtc)
CMD58    sd         READ_EXTR_MULTI (type=adtc)   ;SPI: READ_OCR
CMD59    sd         WRITE_EXTR_MULTI (type=adtc)  ;SPI: CRC_ON_OFF
```

MMC Data Streaming Commands (class 1/class 3)

```
CMD11    mmc        READ_DAT_UNTIL_STOP (class 1) (type=adtc)
CMD20    mmc        WRITE_DAT_UNTIL_STOP (class 3) (type=adtc)
```

Below CMD58-59 SPI-only (in Non-SPI mode: MMC=Reserved, SD=EXTR_MULTI)

```
CMD58    sd/mmc+spi READ_OCR     ;SPI-only ;SD Mode: READ_EXTR_MULTI
CMD59    sd/mmc+spi CRC_ON_OFF   ;SPI-only ;SD Mode: WRITE_EXTR_MULTI
```

Above two commands are supported in SPI mode only, and are supported for both SD and MMC (though newer MMC docs are no longer mentioning them since JEDEC dropped SPI support).

Application Specific Commands (prefixed by CMD55 aka APP_CMD)

```
ACMD6     sd        SET_BUS_WIDTH (type=ac)
ACMD13    sd    spi SD_STATUS (type=adtc) (get 512bit SSR)
ACMD22    sd    spi GET_NUM_WR_BLOCKS (type=adtc)
ACMD23    sd    spi SET_WR_BLK_ERASE_COUNT (type=ac)
ACMD41    sd    spi SD_SEND_OP_COND (type=bcr)  ;SPI: reduced functionality
ACMD42    sd    spi SET_CLR_CARD_DETECT (type=ac)
ACMD51    sd    spi GET_SCR (type=adtc)
ACMD1-5   -         Reserved
ACMD7-12  -         Reserved
ACMD14-16 sd        Reserved for DPS Specification (Data Protection System)
ACMD17    -         Reserved
```

```
ACMD18    sd    spi Reserved for SD security applications
ACMD19-21 -         Reserved
ACMD24    -         Reserved
ACMD25    sd    spi Reserved for SD security applications
ACMD26    sd    spi Reserved for SD security applications
ACMD27    -         Shall not use this command
ACMD28    sd        Reserved for DPS Specification (Data Protection System)
ACMD29    -         Reserved
ACMD30-35 sd        Reserved for Security Specification
ACMD36-37 -         Reserved
ACMD38    sd    spi Reserved for SD security applications
ACMD39-40 -         Reserved
ACMD43-49 sd    spi Reserved for SD security applications
ACMD52-54 sd        Reserved for Security Specification
ACMD55    -         Not exist (equivalent to CMD55)
ACMD56-59 sd        Reserved for Security Specification
ACMD0     -         Unknown/Unused/Undocumented
ACMD50    -         Unknown/Unused/Undocumented
ACMD60-63 -         Unknown/Unused/Undocumented
```

Card Registers

```
CSR       32bit  sd/mmc spi Card Status: command error & state information
OCR       32bit  sd/mmc spi Operation Conditions Register
CID      128bit  sd/mmc spi Card Identification
CSD      128bit  sd/mmc spi Card-Specific Data (CSD Version 1.0 and 2.0)
RCA       16bit  sd/mmc     Relative Card Address (not used in SPI mode)
DSR       16bit  sd/mmc spi Driver Stage Register (optional)
SSR      512bit  sd     spi SD Card Status Register: Extended status field
SCR       64bit  sd     spi SD Card Configuration Register
EXT_CSD 4096bit  mmc    spi MMC Extended CSD Register (status & config)
PWD      128bit  sd/mmc spi Password (Card Lock) (max 16 bytes)
PWD_LEN    8bit  sd/mmc spi Password Length (0..16 max) (0=no password)
```

SD Mode Response Types

```
N/A    0bit  CMD0, CMD4, CMD15   No response
R1    48bit  Normal CMDs/ACMDs   32bit CSR Card Status
R1b   48bit  Busy CMDs/ACMDs     32bit CSR Card Status (and DATA=busy)
R2   136bit  CMD9                120bit CSD Card-Specific Data
R2   136bit  CMD2, CMD10         120bit CID Card Identification
R3    48bit  ACMD41, MMC:CMD1    32bit OCR Register (without crc7)
R4        -  -                   Reserved for SDIO
R5        -  -                   Reserved for SDIO
R6    48bit  CMD3                16bit RCA and cut-down 16bit CSR
R7    48bit  CMD8                32bit Card interface condition
```

SPI Mode Response Types

```
R1     8bit  Normal CMDs/ACMDs   8bit CSR Card Status
R1b    8bit  Busy CMDs/ACMDs     8bit CSR Card Status (and DATA=busy)
R2    16bit  CMD13, ACMD13       16bit CSR Card Status
R3    40bit  CMD58               8bit CSR and 32bit OCR
R4        -  -                   Reserved for SDIO
R5        -  -                   Reserved for SDIO
R6        -  -                   Reserved
R7    40bit  CMD8                8bit CSR and 32bit Card interface condition
ERROR  8bit  Only first 8bit sent upon Illegal Command or Command CRC Error
```

Commands with Data Transfers (additionally to command/response) (type=adtc)

```
CMD17,18 R   sd/mmc spi READ_SINGLE_BLOCK, READ_MULTIPLE_BLOCK
CMD24,25 W   sd/mmc spi WRITE_BLOCK, WRITE_MULTIPLE_BLOCK
CMD8     R   mmc    spi GET_EXT_CSD (4096bit)
CMD9     R   sd/mmc spi GET_CSD (128bit)  ;\in SPI Mode only (Non-SPI mode
CMD10    R   sd/mmc spi GET_CID (128bit)  ;/sends that info as CMD response)
ACMD13   R   sd     spi SD_STATUS (512bit SSR register)
```

```
 ACMD22   R   sd     spi GET_NUM_WR_BLOCKS (32bit counter)
 ACMD51   R   sd     spi GET_SCR (64bit SCR register)
 CMD14,19 R/W mmc        BUSTEST_R, BUSTEST_W
 CMD19    W?  sd         SET_TUNING_BLOCK (512bit tuning pattern)
 CMD27    W   sd/mmc spi PROGRAM_CSD (128bit CSD register)
 CMD30    R   sd/mmc spi GET_WRITE_PROT (32bit write-protect flags)
 CMD31    R   mmc        GET_WRITE_PROT_TYPE (32x2bit write-protect types)
 CMD42    W   sd/mmc spi LOCK_UNLOCK (password header/data)
 CMD6     ??  sd     spi SWITCH_FUNC
 CMD40    ?   sd         Defined by DPS Spec (Data Protection System)
 CMD48,49 R/W sd         READ_EXTR_SINGLE, WRITE_EXTR_SINGLE
 CMD58,59 R/W sd         READ_EXTR_MULTI, WRITE_EXTR_MULTI
 CMD56    R/W sd     spi GEN_CMD
 CMD11    R   mmc        READ_DAT_UNTIL_STOP (class 1) (type=adtc)
 CMD20    W   mmc        WRITE_DAT_UNTIL_STOP (class 3) (type=adtc)
xR1b      R   sd/mmc spi Busy signal for commands with "R1b" response
```

Misnamed Commands

Official command names include various SEND_xxx commands, which are misleading because they don't indicate if they "send" information <to> or <from> the card (or both). Better naming would be GET_xxx, SET_xxx, or GET_SET_xxx.

```
 Official Name         Renamed
 ALL_SEND_CID          ALL_GET_CID
 SEND_CID              GET_CID
 SEND_CSD              GET_CSD
 SEND_STATUS           GET_STATUS
 SEND_RELATIVE_ADDR    GET_RELATIVE_ADDR
 SEND_SCR              GET_SCR
 SEND_EXT_CSD          GET_EXT_CSD
 SEND_WRITE_PROT       GET_WRITE_PROT
 SEND_WRITE_PROT_TYPE  GET_WRITE_PROT_TYPE
 SEND_NUM_WR_BLOCKS    GET_NUM_WR_BLOCKS
 SEND_IF_COND          SET_IF_COND             ;-to card
 SEND_TUNING_BLOCK     SET_TUNING_BLOCK        ;-to card
 SEND_OP_COND          ...
 SD_SEND_OP_COND       ...
```

Other misnamed commands include SET_BLOCKLEN occassionally spelled SET_BLOCK_LEN in SD specs. SELECT_DESELECT_CARD is officially spelled SELECT/DESELECT_CARD.

Difference of SD Commands Definition in UHS-II

SD-TRAN driver of host should manage the difference of SD commands functions. Not supported commands should not issue to UHS-II card. CMD13 shall not be issued during data transfer. Normally, data transfer should be stopped by setting TLEN instead of using CMD12.

CMD23 and CMD55 functions are included in UHS-II packet functions.

```
CMD0   Terminate SD transaction and reset SD-TRAN state.
CMD3   Returns Device ID in the response instead of RCA
CMD4   Illegal
CMD6   Function Group 1 and 3 are not used.
CMD7   Device ID is set to the argument instead of RCA
CMD13  Device operation is up to implementation during data transfer (eg. CTS)
CMD11  Illegal
CMD12  Normally, TLEN (data length) in UHS-II packet is used to stop data
       transfer.
CMD12  Should be used to abort an operation when illegal situation occurs.
CMD15  Illegal
CMD19  Illegal
CMD23  Not Affected. TLEN in UHS-II packet is used to specify data length.
CMD55  Not Affected. ACMD is set by APP field in UHS-II packet.
ACMD6  Illegal
ACMD42 Illegal
```

Not Affected means that the command is not executed in any card state, and response is returned (response type is up to implementation).

Illegal means that card returns response with NACK=1.
As SDHC/SDXC Cards do not support CMD28, 29 and 30, these commands are also illegal in UHS-II mode.

Note
All future reserved commands shall have a codeword length of 48 bits, as well as their responses (if there are any).

DSi SD/MMC Protocol: General Commands

CMD0 - SD/MMC - SPI - GO_IDLE_STATE (type=bc)
Parameter bits:
```
  31-0  stuff bits
```
SD Mode Response: N/A
SPI Mode Response: R1
Resets all cards to idle state, it's usually sent to (re-)invoke card detection and initialization. The command does also seem to reset many further registers (for example, TRAN_SPEED is said to be reset to 25MHz, and, although not officially specified, the DSi's eMMC chip appears to get forced back to 1bit data bus mode). Observe that card detection/initialization should be done at lower CLK rate than usually (MMC specifies max 400kHz - this is actually required - the DSi's onboard Samsung KMAPF0000M-S998 eMMC chip won't respond to ALL_GET_CID when trying to use 16MHz CLK), higher CLK can be used once when detecting max speed (TRAN_SPEED in CSD register).
The command is also used to enter SPI mode (in SPI mode, the /CS pin is held low, while in 1bit/4bit mode that pin would be DAT3=floating/high), SPI commands can be sent without CRCs, however, at time when entering SPI mode, memory cards may still insist on checksums, CMD0 should be thus always sent with CRC7.

CMD8 - SD (SD v2.00 and up) - SPI - SET_IF_COND (type=bcr)
Parameter bits:
```
  31-12 reserved bits
  11-8  supply voltage (VHS)
  7-0   check pattern
```
SD Mode Response: R7:
```
  47      Start Bit (0)                              ;\
  46      Transmission To Host (0)                   ; 1st byte
  45-40   Command (the 6bit CMD being responded to)  ;/
  39-20   Reserved (zero filled)          (20bit)    ;\2nd..4th byte
  19-16   Voltage accepted (see below)    (4bit)     ;/
  23-8    Echo-back of check pattern      (8bit)     ;-5th byte
  7-1     CRC7                                       ;\6th byte
  0       End Bit (1)                                ;/
```
SPI Mode Response: R7:
```
  39-32  R1 (8bit Card Status, same as in normal SPI command responses)
  31-28  Command version (???)         (4bit)
  27-12  Reserved (0)                  (16bit)
  11-8   Voltage Accepted (see below)  (4bit)
  7-0    Echo-back of check pattern    (8bit)
```
Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage.
Voltage Accepted values:
```
  0001b  = 2.7-3.6V
  0010b  = Reserved for Low Voltage Range
  0100b  = Reserved
  1000b  = Reserved
  Others = Not Defined
```
The card supported voltage information of 3.3V range power pin is sent by the response of CMD8. Bits 19-16 indicate the voltage range that the card supports. The card that accepted the supplied voltage returns R7 response. In the response, the card echoes back both the voltage range and check pattern set in the argument.

CMD11 - SD - VOLTAGE_SWITCH (type=ac)
Parameter bits:
```
  31-0  reserved bits (0)
```
Response: R1
Switch to 1.8V bus signaling level.

CMD12 - SD/MMC - SPI - STOP_TRANSMISSION (type=ac)
Parameter bits:
```
  31-0  stuff bits
```
Response: R1b
Additional Data Transfer (from card): Busy signal for "R1b" response
Forces the card to stop transmission (SPI: in Multiple Block Read Operation).
Note: Toshiba SD/MMC controllers are sending STOP_TRANSMISSION automatically.

CMD15 - SD/MMC - GO_INACTIVE_STATE (type=ac)
Parameter bits:
```
  31-16 RCA
  15-0  reserved bits (0)
```
Response: N/A
Sends an addressed card into the Inactive State. This command is used when the host explicitly wants to deactivate a card once and forever (and won't even react to GO_IDLE_STATE) until next power-up (aka until ejecting/reinserting the card).

CMD59 - SD/MMC - SPI-ONLY (not Non-SPI Mode) - CRC_ON_OFF
Supported in SPI Mode only (in Non-SPI mode, CMD59 would be: MMC=Reserved, SD=WRITE_EXTR_MULTI)!
Parameter bits:
```
  31-1  stuff bits
  0     CRC option (0=off, 1=on)
```
SPI Mode Response: R1
Default on power up is unknown. Also unknown if this does completely prevent transmission of both CRC7 and CRC-CCITT values (especially in case of CID/CSD registers that have the CRC7 "inside" of the "128bit" register). Also unknown if CID/CSD are having "double" checksums (CRC7 plus CRC-CCITT) when transferring them as DATA packet (instead of as normal command/response).

ACMD6 - SD - SET_BUS_WIDTH (type=ac)
```
  31-2  stuff bits
  1-0   Bus width for Data transfers (0=1bit, 2=4bit, 1/3=reserved).
```
Response: R1
The supported widths can be found in SCR register. The current width is stored in SSR register. Default width is 1bit on power up.
Note: MMC uses a different mechanism to change the bus-width (via EXT_CSD).

ACMD42 - SD - SPI - SET_CLR_CARD_DETECT (type=ac)
```
  31-1  stuff bits
  0     set_cd (0=Disconnect, 1=Connect)
```
Response: R1
Connect/Disconnect the 50 KOhm pull-up resistor on CD/DAT3 pin of the card.
The pull-up might be intended for card detection (other than by using the slot's card detect switch), and/or for sensing SPI mode (which would drag that pin to LOW level when asserting /CS chip select).
During operation, disabling the pull-up might improve 4bit mode data transfers (unless for card controllers which do rely on the card pull-up to be present). The TC6387XB datasheet recommends external 100K pull-ups on DAT0-2, and only 47K on DAT3 (not quite sure why, unless Toshiba believed the parallel 50K+47K pull-ups to sum up to 100K, rather than to 25K).

CMD55 - SD/MMC (MMC: only in SPI-mode?) - SPI - APP_CMD (type=ac)

```
  31-16  RCA (SPI Mode: stuff bits)
  15-0   stuff bits
```

Response: R1

Used as prefix for application specific commands, ie. the next command will be treated as "ACMDnn" rather than as normal "CMDnn".

As the name says, this was originally intended for "application specific" extensions, however, in the SD Card protocol, it's also used for some ACMD's that are part of the SD protocol.

CMD56 - SD/MMC (MMC: only in SPI-mode?) - SPI - GEN_CMD (type=adtc)

```
  31-1  stuff bits
  0     RD/WR Direction (0=Write to Card, 1=Read from Card)
```

Response: R1

Additional Data Transfer (to/from card, depending on above R/W bit):

```
  General purpose data
  For SDSC, block length is set via SET_BLOCKLEN command.
  For SDHC/SDXC, block length is fixed to 512 bytes.
```

Used to transfer a data block to/from the card for general purpose/application specific commands.

CMD14 - MMC - BUSTEST_R (type=adtc) (MMC only, Reserved on SD)
CMD19 - MMC - BUSTEST_W (type=adtc) (MMC only, SET_TUNING_BLOCK on SD)

```
  31-0  stuff bits
```

Response: R1

Additional Data Transfer (to/from card):

```
  test pattern (2bit per DATA line? eg. 8bit pattern in 4bit-mode?)
```

MMC only. And, that, in Non-SPI mode only.

BUSTEST_W: Host sends the "bus TEST Data pattern" to card.

BUSTEST_R: Host reads the "REVERSED bus TESTING data pattern" from card.

The reversing is said to change a 2bit value of "01" into "10", unknown if that means that the bit-order is reversed, or that the bits are inverted.

DSi SD/MMC Protocol: Block Read/Write Commands

CMD16 - SD/MMC - SPI - SET_BLOCKLEN (type=ac)

```
  31-0  Block length (for Block Read, Block Write, Lock, and GEN_CMD)
```

Response: R1

In the case of SDSC Card, this command sets the block length (in bytes) for all following block commands (read, write, lock). Default block length is fixed to 512 Bytes. Set length is valid for memory access commands only if partial block read operation are allowed in CSD.

In the case of SDHC/SDXC Cards, block length set by CMD16 command does not affect memory read and write commands. Always 512 Bytes fixed block length is used. This command is effective for LOCK_UNLOCK command.

In both cases, if block length is set larger than 512 Bytes, the card sets the BLOCK_LEN_ERROR bit.

In DDR50 mode, block length must be even (because data is sampled on both clock edges).

CMD20 - SD (optional, see SCR.Bit32) - SPEED_CLASS_CONTROL (type=ac)

```
  31-28  Speed Class Control (for Block Read, and Block Write commands)
  27-0   Reserved (0)
```

Response: R1b

Additional Data Transfer (from card): Busy signal for "R1b" response

Speed Class control command. Refer to Section 4.13.2.8.

CMD23 - SD/MMC - SPI (but only on MMC) - SET_BLOCK_COUNT (type=ac)

Supported by SD and MMC Cards. However, in SPI-mode it's supported only for MMC? And, for SD it's optional (see SCR.Bit33).

```
  31-0  Block Count (MMC: only lower 16bit used, upper 16bit=reserved)
```

Response: R1
Specify block count for CMD18 and CMD25.

```
 _____________________ Block-Oriented READ Commands _____________________
```

CMD17 - SD/MMC - SPI - READ_SINGLE_BLOCK (type=adtc)

```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```

Response: R1
Additional Data Transfer (from card):

```
  data
```

In the case of SDSC Card, this command reads a block of the size selected by the SET_BLOCKLEN command. The data transferred shall not cross a physical block boundary unless READ_BLK_MISALIGN is set in the CSD.
In case of SDHC and SDXC Cards, block length is fixed 512 Bytes regardless of the SET_BLOCKLEN command.

CMD18 - SD/MMC - SPI - READ_MULTIPLE_BLOCK (type=adtc)

```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```

Response: R1
Additional Data Transfer (from card):

```
  data
```

Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command. Block length is specified the same as READ_SINGLE_BLOCK command.

CMD19 - SD - SET_TUNING_BLOCK (type=adtc) (SD only, BUSTEST_W on MMC)

```
  31-0  reserved bits (0)
```

Response: R1
Additional Data Transfer (to card):

```
  64 bytes (512bit) tuning pattern is sent for SDR50 and SDR104.
```

```
 _____________________ Block-Oriented WRITE Commands _____________________
```

CMD24 - SD/MMC - SPI - WRITE_BLOCK (type=adtc)

```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```

Response: R1
Additional Data Transfer (to card):

```
  data
```

In case of SDSC Card, block length is set by the SET_BLOCKLEN command. The data transferred shall not cross a physical block boundary unless WRITE_BLK_MISALIGN is set in the CSD. In the case that write partial blocks is not supported, then the block length=default block length (given in CSD).
In case of SDHC and SDXC Cards, block length is fixed 512 Bytes regardless of the SET_BLOCKLEN command.

CMD25 - SD/MMC - SPI - WRITE_MULTIPLE_BLOCK (type=adtc)

```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```

Response: R1
Additional Data Transfer (to card):

```
  data
```

Continuously writes blocks of data until a STOP_TRANSMISSION follows.
Block length is specified the same as WRITE_BLOCK command.

ACMD22 - SD - SPI - GET_NUM_WR_BLOCKS (type=adtc)

```
  31-0  stuff bits
```

Response: R1
Additional Data Transfer (from card):
```
  31-0  Number of the written (without errors) write blocks (32bit)
```
Responds with 32bit+CRC data block.
If WRITE_BL_PARTIAL='0', the unit of ACMD22 is always 512 byte.
If WRITE_BL_PARTIAL='1', the unit of ACMD22 is a block length which was used when the write command was executed.

ACMD23 - SD - SPI - SET_WR_BLK_ERASE_COUNT (type=ac)
```
  31-23  stuff bits
  22-0   Number of blocks
```
Response: R1
Set the number of write blocks to be pre-erased before writing (to be used for faster Multiple Block WR command). "1"=default (one wr block).
Command STOP_TRAN (CMD12) shall be used to stop the transmission in Write Multiple Block whether or not the pre-erase (ACMD23) feature is used.

```
_____________________ Byte-Streaming READ/WRITE Commands _____________________
```

CMD11 - MMC - READ_DAT_UNTIL_STOP (class 1) (type=adtc)
CMD20 - MMC - WRITE_DAT_UNTIL_STOP (class 3) (type=adtc)
```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```
Response: R1
Additional Data Transfer (to/from card):
```
  data
```
Similar to read/write multiple blocks, but transferring the data as an endless byte stream (instead of splitting it into separate blocks). Transfer is terminated by sending STOP_TRANSMISSION.

DSi SD/MMC Protocol: Special Extra Commands

```
_____________________________ Write PROTECTION Commands _____________________________
```

CMD28 - SDSC/MMC (not SDHC/SDXC) - SPI - SET_WRITE_PROT (type=ac)
CMD29 - SDSC/MMC (not SDHC/SDXC) - SPI - CLR_WRITE_PROT (type=ac)
```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: Unsupported)
```
Response: R1b
Additional Data Transfer (from card): Busy signal for "R1b" response
Write protection support is indicated in CSD(WP_GRP_ENABLE), and additionally "class 6" should be flagged in CSD(CCC). The group size is indicated in CSD(WP_GRP_SIZE), observe that that field is 5bit/7bit wide for SD/MMC accordingly.

CMD30 - SDSC/MMC (not SDHC/SDXC) - SPI - GET_WRITE_PROT (type=adtc)
```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: Unsupported)
```
Response: R1
Additional Data Transfer (from card):
```
  31-0  Flags (1=write-protected) (bit0=addressed group, bit1..31=next groups)
```
If the card provides write protection features, this command asks the card to send the status of the write protection bits: 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits are transferred in a payload format via the DATA line. The last (least significant) bit of the protection bits corresponds to the first addressed group. If the addresses of the last groups are outside the valid range, then the corresponding write protection bits shall be set to 0.

CMD31 - MMC - GET_WRITE_PROT_TYPE (type=adtc)
```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: Unsupported)
```

Response: R1
Additional Data Transfer (from card):
```
  63-0  Flags (1=write-protected) (bit0-1=addressed group, bit2..63=next)
```
Returns thirty-two 2bit values (0=not protected, 1=temporary write protection, 2=power-on write protection, 3=permanent write protection).

Further Write-Protection Mechanisms
The whole card can be write-protected via PERM_WRITE_PROTECT and TMP_WRITE_PROTECT bits in CSD register (supported for MMC and SDSC/SDHC/SDXC).
SD Cards (and SD Card adaptors for miniSD and microSD cards) are additionally having a mechanical "LOCK" write protection tab (MMC cards don't have that feature).
The PWD feature provides Read/Write-protection (when not knowing the password).

_____ Erase Commands _____

CMD32 - SD - SPI - ERASE_WR_BLK_START (type=ac)
CMD33 - SD - SPI - ERASE_WR_BLK_END (type=ac)
```
  31-0  data address (SDSC: in 1-byte units, SDHC/SDXC: in 512-byte units)
```
Response: R1
Sets the address of the first/last write block of the continuous range to be erased.

CMD35 - MMC - SPI - ERASE_GROUP_START (type=ac)
CMD36 - MMC - SPI - ERASE_GROUP_END (type=ac)
```
  31-0  data address (MMC: in WHAT units?)
```
Response: R1
MMC only. Unknown, maybe similar to above SD commands?

CMD32-34,37 - SPI - MMC - Reserved for compatibility with older MMC cards
MMC only. Unknown, maybe also Erase related?

CMD38 - SD/MMC - SPI - ERASE (type=ac)
```
  31-0  stuff bits
```
Response: R1b
Additional Data Transfer (from card): Busy signal for "R1b" response
Erases all previously selected write blocks.

Further Erase Commands
Sectors are automatically erased on-the-fly when writing data blocks, so manually using above erase commands isn't really necessary; it may be useful for shreddering private data though, and it might also speed up subsequent writes since the writes can omit the on-the-fly erasing step.
The SET_WR_BLK_ERASE_COUNT (ACMD23) can be used to notify the card that it may pre-erase multiple sectors upon write commands (eg. to speed-up cluster writes that are spanning across multiple sectors).
The password lock feature includes a Forced Erase function, which will reset the password, and ERASE THE WHOLE CARD, this may be useful if the user has forgot the password, but will destroy data (possibly including the pre-formatted filesystem headers; which would be bad, because that headers should contain cluster sizes somewhat matched to the physical sector sizes).

_____ I/O Commands _____

CMD5 - SD - SPI - Reserved for I/O cards
CMD52-54 - SD - SPI - Commands for SDIO
```
  CMD5  SDIO: IO_SEND_OP_COND
  CMD52 SDIO: IO_RW_DIRECT
  CMD53 SDIO: IO_RW_EXTENDED
```

Refer to the "SDIO Card Specification". SDIO is an extension to the SD protocol that allows to access non-memory-card hardware (such like cameras or network adaptors).

Switch Function Commands

CMD6 - SD (SD v1.10 and up) - SPI - SWITCH_FUNC (type=adtc)

```
31     Mode (0=Check function, 1=Switch function)
30-24  reserved (All '0')
23-20  function group 6: Reserved (0h or Fh)
19-16  function group 5: Reserved (0h or Fh)
15-12  function group 4: Power Limit     ;SPI Mode: Reserved (0h or Fh)
11-8   function group 3: Drive Strength  ;SPI Mode: Reserved (0h or Fh)
7-4    function group 2: Command System
3-0    function group 1: Access Mode
```

Response: R1
Additional Data Transfer (to/from whatever):

```
unknown
```

Checks switch-able function (mode 0) and switch card function (mode 1). See Chapter 4.3.10.

CMD34-35 - SD - Reserved
Reserved for each command system set by switch function command (CMD6).
Detailed definition is referred to each command system specification.
Maybe related to above "function group 5..6"?

CMD36,37 - SD - Undoc (description field is held blank)
CMD50,57 - SD - Undoc (description field is held blank)
Undoc. Maybe related to above "function group 1..4"?

SPI: CMD34-37 - SD - SPI - Reserved for Command Systems from CMD6
SPI: CMD50,57 - SD - SPI - Reserved for Command Systems from CMD6
Described as so for SPI Mode. Maybe related to above "function group 1..6"?

Function Extension Commands

CMD21 - SD - Reserved for DPS Specification (Data Protection System)
Reserved.

CMD48 - SD (optional, see SCR.Bit34) - READ_EXTR_SINGLE (type=adtc)

```
31     MIO (0=Memory, 1=I/O)
30-27  FNO
26     Reserved (0)
25-9   ADDR
8-0    LEN
```

Response: R1
Additional Data Transfer (from card):

```
whatever
```

Single block read type. Refer to Section 5.7.2.1.

CMD49 - SD (optional, see SCR.Bit34) - WRITE_EXTR_SINGLE (type=adtc)

```
31     MIO (0=Memory, 1=I/O)
30-27  FNO
26     MW
25-9   ADDR
8-0    LEN/MASK
```

Response: R1
Additional Data Transfer (to card):

```
whatever
```

Single block write type. Refer to Section 5.7.2.2.

CMD58 - SD (optional, see SCR.Bit35) - READ_EXTR_MULTI (type=adtc)

```
  31     MIO (0=Memory, 1=I/O)
  30-27  FNO
  26     BUS (0=512B, 1=32KB)
  25-9   ADDR
  8-0    BUC
```

Response: R1
Additional Data Transfer (from card):

```
  whatever
```

Multi-block read type. Refer to Section 5.7.2.4.

CMD59 - SD (optional, see SCR.Bit35) - WRITE_EXTR_MULTI (type=adtc)

```
  31     MIO (0=Memory, 1=I/O)
  30-27  FNO
  26     BUS (0=512B, 1=32KB)
  25-9   ADDR
  8-0    BUC
```

Response: R1
Additional Data Transfer (to card):

```
  whatever
```

Multi-block write type. Refer to Section 5.7.2.5.

Note: CCC bit 11 is set to 1 when any command of class 11 is supported. Supporting of these commands is indicated in SCR register.

DSi SD/MMC Protocol: CSR Register (32bit Card Status Register)

CMD13 - SD/MMC - SPI - GET_STATUS (type=ac)
Parameter bits:

```
  31-16 RCA (SPI Mode: stuff bits)
  15-0  stuff bits
```

SD Mode Response: R1 (32bit Card Status):

```
  47      Start Bit (0)                                ;\
  46      Transmission To Host (0)                     ; 1st byte
  45-40   Command (the 6bit CMD being responded to)    ;/
  39-8    CSR Card Status Register (32bit) (see below) ;-2nd..5th byte
  7-1     CRC7                                         ;\6th byte
  0       End Bit (1)                                  ;/
```

SPI Mode Response: R2 (16bit Card Status):

```
  15-0    CSR Card Status Register (16bit) (see below)  ;-1st..2nd byte
```

Addressed card sends its status register.

CMDxx/ACMDxx - Other Commands
Most other commands are also returning the Card Status in their responses:
SD Mode Response: R1 (32bit Card Status)
SPI Mode Response: R1 (8bit Card Status; most SPI commands return only 8bit)
SPI Mode Response: R2 (16bit Card Status; SPI commands CMD13/ACMD return 16bit)

CMDxx/ACMDxx - Other Commands with R1b Response
R1b is identical to R1, with an optional busy signal transmitted on the DATA line (R1b occurs for CMD7, CMD12, CMD20, CMD28, CMD29, CMD38) (and for MMC: also for CMD5, CMD6). The card may become busy after receiving these commands based on its state prior to the command reception. The Host shall check for busy at the response.
In SD Mode, the busy signal is sent on DAT0 line (DAT1-3 aren't used, even if the card is in 4bit mode). The

busy signal does consist of BITs? (not bytes?), and has a "start bit"?, followed by what-value-when-busy? and what-final-value-when-ready?
In SPI Mode, the busy signal is sent as BYTEs (00h=Busy, xxh=Nonzero=Ready).

CSR Card Status Register (full 32bit, as returned in SD Mode Response: R1)

Bit	Typ	Clr	Identifier	Meaning
31	ERX	C	OUT_OF_RANGE	(1=Command's argument was out of range)
30	ERX	C	ADDRESS_ERROR	(1=Misaligned address/block len mismatch)
29	ERX	C	BLOCK_LEN_ERROR	(1=Wrong block length, bytelen mismatch)
28	ER	C	ERASE_SEQ_ERROR	(1=Error in erase command sequence)
27	ERX	C	ERASE_PARAM	(1=Wrong erease selection of write-blocks)
26	ERX	C	WP_VIOLATION	(1=Write failed due to write-protection)
25	SX	A	CARD_IS_LOCKED	(1=Card is locked by the host)
24	ERX	C	LOCK_UNLOCK_FAILED	(1=Lock/unlock sequence or password error)
23	ER	B	COM_CRC_ERROR	(1=CRC check of previous command failed)
22	ER	B	ILLEGAL_COMMAND	(1=Command not legal for the card state)
21	ERX	C	CARD_ECC_FAILED	(1=Internal error correction failed)
20	ERX	C	CC_ERROR	(1=Internal card controller error)
19	ERX	C	ERROR	(1=General error, or Unknown error)
18	-	-	Reserved (eMMC: UNDERRUN)	
17	-	-	Reserved (eMMC: OVERRUN) (eSD: DEFERRED_RESPONSE)	
16	ERX	C	CSD_OVERWRITE	(1=read-only CSD section doesn't match card content, or attempted to reverse the Copy/WP bits)
15	ERX	C	WP_ERASE_SKIP	(1=partial erase error due to write-protect)
14	SX	A	CARD_ECC_DISABLED	(1=Internal error correction wasn't used)
13	SR	C	ERASE_RESET	(1=Erase sequence was aborted)
12-9	SX	B	CURRENT_STATE	(00h..0Fh=state, see below)
8	SX	A	READY_FOR_DATA	(1=Ready/buffer is empty)
7	EX	C	SWITCH_ERROR	(1=SWITCH command refused, MMC only)
6	-	-	Reserved/Unspecified (description is left blank)	
5	SR	C	APP_CMD	(1=Card will expect ACMD)
4	-	-	Reserved for SD I/O Card	
3	ER	C	AKE_SEQ_ERROR	(1=Authentication Sequence Error)
2	-	-	Reserved for application specific commands	
1-0	-	-	Reserved for manufacturer test mode	

Values for CURRENT_STATE (bit12-9):

```
These bits indicate the OLD state of card when receiving the command,
(ie. if the command does change the state, then the NEW state won't be
seen until the NEXT command returns the new updated status bits)
00h     = idle
01h     = ready
02h     = ident
03h     = stby
04h     = tran    ;<-- normal state (when waiting for read/write commands)
05h     = data    ;data read  (CMD8,CMD11,CMD17,CMD18,CMD30,CMD56/R)
06h     = rcv     ;data write (CMD20?,CMD24,CMD25,CMD26,CMD27,CMD42,CMD56/W)
07h     = prg     ;erase/wprot (CMD6,CMD28,CMD29,CMD38)
08h     = dis
09h     = btst    ;bus test write (CMD19, MMC only)
0Ah     = slp     ;sleep (CMD5, MMC only)
0Bh-0Eh = reserved
0Fh     = reserved for I/O mode (SDIO-only devices, without SD-memory)
N/A     = ina     ;inactive (CMD15) (card is killed, and can't send status)
N/A     = irq     ;interrupt mode (CMD40, MMC only)
N/A     = pre     ;pre-idle (MMC only)
```

Type aka Typ column (in above table):

```
E: Error bit.
S: Status bit.
R: Flag may get set within response of current command.
X: Flag may get set within response of NEXT command (with R1 response)
```

Clear Condition aka Clr column (in above table):

```
A: According to the card current state.
```

```
  B: Always related to the previous command. Reception of a valid command
     will clear it (with a delay of one command).
  C: Clear by read.
```

SPI Responses (8bit "R1" Responses, and 16bit "R2" Responses)

```
 FIRST BYTE of all SPI Responses:
  7  always 0                          ;\
  6  parameter error                   ; These 8bit are returned in ALL normal
  5  address error                     ; SPI commands (with 8bit "R1" response)
  4  erase sequence error              ; and,
  3  com crc error                     ; the same 8bits are also returned
  2  illegal command                   ; as FIRST BYTE in SPI commands with
  1  erase reset                       ; longer responses
  0  in idle state                     ;/
 SECOND BYTE of SPI "R2" Response:
  7  out of range, or csd overwrite           ;\
  6  erase param                              ;
  5  wp violation                             ; These extra 8bits are returned
  4  card ecc failed                          ; as SECOND BYTE in SPI commands
  3  CC error                                 ; with 16bit "R2" status response
  2  error                                    ; (ie. in CMD13 and ACMD13)
  1  wp erase skip, or lock/unlock cmd failed ;
  0  Card is locked                           ;/
```

Card Status Field/Command - Cross Reference

For each command responded by R1 response, following table defines the affected bits in the status field. An 'x' means the error/status bit may be set in the response to the respective command.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12-9	8	5
CMD3									x	x			x							x		
CMD6	x						x		x	x	x	x	x							x		
CMD7					x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD11							x		x	x			x							x		
CMD12	x	x				x	x		x	x	x	x	x					x		x		
CMD13	x	x			x	x	x	x	x	x	x	x	x			x	x	x		x	x	
CMD16			x		x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD17	x	x			x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD18	x	x			x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD19	x	x			x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD20	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD23	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD24	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD25	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD26					x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD27					x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD28	x				x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD29	x				x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD30	x				x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD32	x			x	x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD33	x			x	x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD38				x	x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD42					x	x	x	x	x	x	x	x	x			x	x	x	x	x		
CMD48	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD49	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD55					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
CMD56					x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	x
CMD58	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
CMD59	x	x	x		x	x	x	x	x	x	x	x	x			x	x	x	x	x	x	
ACMD6	x				x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
ACMD13					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
ACMD22					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
ACMD23					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
ACMD42					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x
ACMD51					x	x	x	x	x	x	x	x	x			x	x	x	x	x		x

Note: The response to CMD3 is R6 that includes only bits 23, 22, 19 and 12:9 out of the Card Status.

DSi SD/MMC Protocol: SSR Register (512bit SD Status Register)

ACMD13 - SD - SPI - SD_STATUS (type=adtc)

```
  31-0   stuff bits
```

SD Mode Response: R1 (32bit Card Status)
SPI Mode Response: R2 (16bit Card Status) (same as for CMD13, see there)
Additional Data Transfer (from card):

```
  511-0  SSR Register (512bit)
```

Send the SD Status. The status fields are given in Table 4-43.

SD Status (transferred on DATA line after ACMD13)

The size of the SD Status is one data block of 512 bit. The content of this register is transmitted to the Host over the DAT bus along with a 16-bit CRC.
ACMD13 can be sent to a card only in 'tran_state' (card is selected).

```
  Bits   Type Clr Identifier
  511-510  SR A   DAT_BUS_WIDTH (0..3, see below)
  509      SR A   SECURED_MODE  (0=Normal, 1=Secured) (Part 3 Security Specs)
  508-502  -  -   Reserved for Security Functions     (Part 3 Security Specs)
  501-496  -  -   Reserved
  495-480  SR A   SD_CARD_TYPE  (0..FFFFh, see below)
  479-448  SR A   SIZE_OF_PROTECTED_AREA   Size of protected area (see below)
  447-440  SR A   SPEED_CLASS      Speed Class of the card (see below)
  439-432  SR A   PERFORMANCE_MOVE Performance of move indicated by 1 MB/s step
  431-428  SR A   AU_SIZE          Size of AU (see below)
  427-424  -  -   Reserved
  423-408  SR A   ERASE_SIZE       Number of AUs to be erased at a time
  407-402  SR A   ERASE_TIMEOUT    Timeout value for erasing areas
                                   specified by UNIT_OF_ERASE_AU (see below)
  401-400  SR A   ERASE_OFFSET     Fixed offset value added to erase time
  399-396  SR A   UHS_SPEED_GRADE  Speed Grade for UHS mode  (see below)
  395-392  SR A   UHS_AU_SIZE      Size of AU  for UHS mode  (see below)
  391-312  -  -   Reserved
  311-0    -  -   Reserved for manufacturer
```

Values for DAT_BUS_WIDTH (as set via SET_BUS_WIDTH command):

```
  00h = 1 bit width (default)
  01h = reserved
  02h = 4 bit width
  03h = reserved
```

Values for SD_CARD_TYPE

```
  0000h = Regular SD RD/WR Card
  0001h = SD ROM Card
  0002h = OTP
  0004h,0008h,0010h,0020h,0040h,0080h = Reserved for future variations
  01xxh..FFxxh = Reserved for Cards that don't comply to Physical Layer Specs
```

Values for SIZE_OF_PROTECTED_AREA

Setting this field differs between SDSC and SDHC/SDXC.
In case of SDSC Card, the capacity of protected area is calculated as follows:

```
  Protected Area = SIZE_OF_PROTECTED_AREA_* MULT * BLOCK_LEN.
  SIZE_OF_PROTECTED_AREA is specified by the unit in MULT*BLOCK_LEN.
```

In case of SDHC and SDXC Cards, the capacity of protected area is calculated as follows:

```
  Protected Area = SIZE_OF_PROTECTED_AREA
  SIZE_OF_PROTECTED_AREA is specified by the unit in byte.
```

Values for SPEED_CLASS

This 8-bit field indicates the Speed Class. Classes lower than indicated by this field are also effective.

00h	Speed Class 0
01h	Speed Class 2
02h	Speed Class 4
03h	Speed Class 6
04h	Speed Class 10
05h-FFh	Reserved for future/faster classes

Application Note:

If a Class value indicated in SD Status (including reserved value) is larger than that of host supported, the host should read as any Class can be used with the card.

For example, Class 10 is indicated, host should consider Class 2 to 6 is also effective.

Values for PERFORMANCE_MOVE

This 8-bit field indicates Pm and the value can be set by 1 [MB/sec] step.

If the card does not move used RUs, Pm should be considered as infinity.

Setting to FFh means infinity.

Pm is defined for Class 2 to 6 in Default Speed Mode. When host uses Class 10, Pm indicated in SD Status shall be ignored and treated as 0.

00h	Sequential Write
01h	1 [MB/sec]
02h	2 [MB/sec]
...	...
FEh	254 [MB/sec]
FFh	Infinity

Values for AU_SIZE

This 4-bit field indicates AU Size and the value can be selected from 16 KB.

00h	Not Defined
01h	16 KB
02h	32 KB
03h	64 KB
04h	128 KB
05h	256 KB
06h	512 KB
07h	1 MB
08h	2 MB
09h	4 MB
0Ah	8 MB
0Bh	12 MB (!)
0Ch	16 MB
0Dh	24 MB (!)
0Eh	32 MB
0Fh	64 MB

Maximum AU size (depending on card capacity):

Card Capacity	up to 64MB	up to 256MB	up to 512MB	up to 32GB	up to 2TB
Maximum AU Size	512 KB	1 MB	2 MB	4 MB1	64MB

The card can set any AU size (up to above maximum AU size values).

The card should set smaller AU size as much as possible.

Application Notes:

The host should determine host buffer size based on total busy time of 4MB and the card supported class. The host can treat multiple AUs combined as one unit.

Values for ERASE_SIZE

This 16-bit field indicates NERASE. When NERASE numbers of AUs are erased, the timeout value is specified by ERASE_TIMEOUT (Refer to ERASE_TIMEOUT).

The host should determine proper number of AUs to be erased in one operation so that the host can indicate progress of erase operation.

```
0000h    Erase Time-out Calculation is not supported.
0001h    1 AU
0002h    2 AU
0003h    3 AU
...      ...
FFFFh    65535 AU
```

Values for ERASE_TIMEOUT

This 6-bit field indicates the TERASE and the value indicates erase timeout from offset when multiple AUs are erased as specified by ERASE_SIZE. The range of ERASE_TIMEOUT can be defined as up to 63 seconds and the card manufacturer can choose any combination of ERASE_SIZE and ERASE_TIMEOUT depending on the implementation. Once ERASE_TIMEOUT is determined, it determines the ERASE_SIZE. The host can determine timeout for any number of AU erase by the Equation (6). Refer to 4.14 for the concept of calculating erase timeout. If ERASE_SIZE field is set to 0, this field shall be set to 0.

```
00h      Erase Time-out Calculation is not supported.
01h      1 [sec]
02h      2 [sec]
03h      3 [sec]
...      ...
3Fh      63 [sec]
```

Values for ERASE_OFFSET

This 2-bit field indicates the TOFFSET and one of four values can be selected. The erase offset adjusts the line by moving in parallel on the upper side. Refer to Figure 4-57 and Equation (6) in 4.14. This field is meaningless if ERASE_SIZE and ERASE_TIMEOUT fields are set to 0.

```
00h      0 [sec]
01h      1 [sec]
02h      2 [sec]
03h      3 [sec]
```

Values for UHS_SPEED_GRADE

This 4-bit field indicates the UHS mode Speed Grade. Reserved values are for future speed grades larger than the highest defined value. Host shall treat reserved values (undefined) as highest grade defined.

```
00h      Less than 10MB/sec
01h      10MB/sec and above
02h-0Fh  Reserved
```

Values for UHS_AU_SIZE

This 4-bit field indicates AU Size for UHS-I and UHS-II cards. Card should set smaller value as much as possible. Host shall refer to UHS_AU_SIZE instead of AU_SIZE when the card is operating in UHS-I or UHS-II bus speed modes.

```
00h      Not Defined
01h-06h  Not Used
07h      1 MB
08h      2 MB
09h      4 MB
0Ah      8 MB
0Bh      12 MB (!)
0Ch      16 MB
0Dh      24 MB (!)
0Eh      32 MB
0Fh      64 MB
```

DSi SD/MMC Protocol: OCR Register (32bit Operation Conditions Register)

CMD1 (MMC) and ACMD41 (SD) are intended to exchange OCR information. That is, the OCR parameter bits

should indicate the host conditions (eg. for DSi: 40100000h, ie. bit20=3.3V supply, and bit30=HCS support for High Capacity carts with more than 2GBytes). The OCR response may then return something like 007f8000h when busy, and 807f8000h when ready (bit20 indicating the voltage being actually supported, bit30 indicating if it's High Capacity card, and bit31 indicating if the card is ready & switched from "idle" to "ready" state).
Cards do usually send at least one response with bit31=0 (busy), one should repeat sending CMD1/ACMD51 until bit31=1 (ready).
Note: All card(s) on the bus will respond to CMD1/ACMD51: with the response bits ANDed together (thus returning nonsense in bit30=HCS when actually sharing the same bus for multiple cards).
Note: The card switches to "ina" state if the voltage in param bits isn't supported.

CMD1 - SD/MMC (For SD Cards: SPI-only) - SPI - SEND_OP_COND
Parameter For MMC Cards (supported in SPI and Non-SPI mode):
```
  31-0  OCR without busy (ie. without the power-up busy flag in bit31)
```
Parameter For SD Cards (supported in SPI mode only, not in Non-SPI mode):
```
  31    Reserved (0)                           ;\special case (applies
  30    HCS (Host Capacity Support information) ; to SD-cards in SPI-mode
  29-0  Reserved (0)                           ;/only)
```
SD Response: R1
MMC Response: R3 (same/similar as SD Mode's ACMD41 response, see below)
Sends host capacity support information and activates the card's initialization process. HCS is effective when card receives SET_IF_COND command.

ACMD41 - SD - SPI - SD_SEND_OP_COND (type=bcr)
```
  31     reserved bit
  30     HCS(OCR[30]) (Host Capacity Support information)
  29     reserved for eSD                 ;\
  28     XPC Max Power Consumption (watts); SPI Mode: Reserved
  27-25  reserved bits                    ; (ie. only bit30 is used for SPI)
  24     S18R                             ; (ie. ACMD41 is SAME as SPI CMD1 ?)
  23-0   VDD Voltage Window(OCR[23-0])    ;/
```
SD Mode Response: R3:
```
  47     Start Bit (0)                                ;\
  46     Transmission To Host (0)                     ; 1st byte
  45-40  Reserved (111111) (instead of Command value) ;/
  39-8   OCR (32bit)                                  ;-2nd..5th byte
  7-1    Reserved (111111) (instead of CRC7)          ;\6th byte
  0      End Bit (1)                                  ;/
```
SPI Mode Response: R1 (without extra Data transfer? use READ_OCR instead?)
Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line. HCS is effective when card receives SET_IF_COND command (uh, but IF_COND should be set BEFORE setting OP_COND?).
Sends request to switch to 1.8V signaling (S18R).
Reserved bit shall be set to '0'. CCS bit is assigned to OCR[30].
XPC controls the maximum power in the default speed mode of SDXC card.
```
  XPC=0: 0.36W (100mA at 3.6V on VDD1) (max) but speed class is not supported.
  XPC=1: 0.54W (150mA at 3.6V on VDD1) (max) and speed class is supported.
```

CMD58 - SD/MMC - SPI-ONLY (not Non-SPI Mode) - READ_OCR
Supported on SD Cards in SPI Mode only (in Non-SPI mode, CMD58 would be: MMC=Reserved, SD=READ_EXTR_MULTI)!
Parameter bits:
```
  31-0  stuff bits
```
SPI Mode Response: R3:
```
  39-32  R1 (8bit Card Status, same as in normal SPI command responses)
  31-0   OCR (32bit)
```

OCR register
The 32-bit operation conditions register stores the VDD voltage profile of the non UHS-II card and VDD1

voltage profile of the UHS-II card. Additionally, this register includes status information bits. One status bit is set if the card power up procedure has been finished. This register includes another status bit indicating the card capacity status after set power up status bit. The OCR register shall be implemented by the cards.
The 32-bit operation conditions register stores the VDD voltage profile of the card.
Bit 7 of OCR is newly defined for Dual Voltage Card and set to 0 in default. If a Dual Voltage Card does not receive CMD8, OCR bit 7 in the response indicates 0, and the Dual Voltage Card which received CMD8, sets this bit to 1.

```
  31     Card power up status bit (0=Busy, 1=Ready)
  30     Card Capacity Status (CCS) (valid only if above Bit31 indicates Ready)
          CCS=0   SDSC Card      (addressed in 1-byte units)   ;MMC max 2GB
          CCS=1   SDHC/SDXC card (addressed in 512-byte units) ;MMC > 2GB
  29     UHS-II Card Status
  28-25  Reserved
  24     Switching to 1.8V Accepted (S18A) (Only UHS-I card supports this bit)
  23     3.5-3.6                          ;\
  22     3.4-3.5                          ;
  21     3.3-3.4                          ;
  20     3.2-3.3 ;<-- this used by DSi    ;
  19     3.1-3.2                          ; VDD Voltage Window
  18     3.0-3.1                          ;
  17     2.9-3.0                          ;
  16     2.8-2.9                          ;
  15     2.7-2.8                          ;
  14-8   Reserved (MMC: 2.0V .. 2.6V)     ;  ;<-- uh, probably in opposite order?
  7      Reserved for Low Voltage Range   ;
  6-4    Reserved                         ;
  3-0    Reserved                         ;/
```

The supported voltage range is coded as shown in Table 5-1. A voltage range is not supported if the corresponding bit value is set to LOW. As long as the card is busy, the corresponding bit (31) is set to LOW.
VDD Voltage Window of OCR indicates VDD1 voltage range in case of UHS-II Card.
UHS-II Card Status bit is added in Bit 29 to indicate whether the card supports UHS-II Interface. Non UHS-II Card sets Bit 29 to 0 and UHS-II Card sets Bit 29 to 1. This bit is not affected by whether VDD2 is supplied or not.

DSi SD/MMC Protocol: CID Register (128bit Card Identification)

CMD2 - SD/MMC - ALL_GET_CID (type=bcr)
CMD2 is/was intended for multiple MMC cards on the same SD/MMC bus, the connected card(s) should compare the CMD2 response bits seen on the bus, and the card with the smallest CID number is switched to "ident" state (and any other cards stay in "ready" state until sending further CMD2's).
CMD2 is still required for both SD and MMC during initialization, although actually sharing the same bus for multiple cards is rather uncommon/depracted (and might envolve various problems: Like conflicting OCR responses, conflicting pull-ups on DAT3 pin, signal noise/spikes on insertion/removal of a second card while accessing another card, problems with (shared) Write Protect and Card Detect switches, and so on).
Parameter bits:

```
  31-0  stuff bits
```

SD Mode Response: R2 (same 136bit response as for CMD10, see there)
Asks any card to send the CID numbers on the CMD line (any card that is connected to the host will respond - until it sees a "0" bit from another card while itself outputting a "1" bit).
Observe that CMD2 (and other card detection/initialization commands) should be done at lower CLK rate than usually (MMC specifies max 400kHz - this is actually required - the DSi's onboard Samsung KMAPF0000M-S998 eMMC chip won't respond to ALL_GET_CID when trying to use 16MHz CLK), higher CLK can be used once when detecting max speed (TRAN_SPEED in CSD register).

CMD10 - SD/MMC - SPI - GET_CID (type=ac)
This command should be used for actually READING the CID (as opposed to ALL_GET_CID which is

primarily intended for the connected card(s) to COMPARE their CIDs with each other).
Parameter bits:

```
  31-16 RCA (SPI Mode: stuff bits)
  15-0  stuff bits
```

SD Mode Response: R2:

```
  135     Start Bit (0)                                ;\
  134     Transmission To Host (0)                     ; 1st byte
  133-128 Reserved (111111) (instead of Command value) ;/
  127-8   CID (120bit) (15 bytes)     ;\aka 128bit     ;-2nd..16th byte
  7-1     CRC7                        ; when including ;\17th byte
  0       End Bit (1)                 ;/CRC7+EndBit    ;/
```

SPI Mode Response: R1, plus DATA line,
SPI Mode Additional Data Transfer (from card):

```
  127-0   CID (128bit) ... or 120bit ?
```

Addressed card sends its card identification (CID).

CID register
The Card IDentification (CID) register is 128 bits wide. It contains the card identification information used during the card identification phase. Every individual Read/Write (RW) card shall have a unique identification number.
For SD Cards (short product name, but bigger date field, 2000..2255?):

```
  Bit     Siz  Field  Name
  127-120   8  MID    Manufacturer ID    (binary) ;\assigned by SD-3C, LLC
  119-104  16  OID    OEM/Application ID (ASCII)  ;/
  103-64   40  PNM    Product name       (ASCII)
  63-56     8  PRV    Product revision   (BCD, 00h-99h) (eg 62h = rev 6.2)
  55-24    32  PSN    Product serial number (32bit)
  23-20     4  -      Reserved (zero)
  19-8     12  MDT    Manufacturing date (yymh) (m=1..12, yy=0..255?; +2000)
  7-1       7  CRC    CRC7 checksum
  0         1  1      Stop bit (always 1)
```

For MMC Cards (smaller date field, range 1997..2012 only):

```
  Bit     Siz  Field  Name
  127-120   8  MID    Manufacturer ID    (binary)   ;\assigned by MMCA
  119-104  16  OID    OEM/Application ID (binary)   ;/  ... or ...
  127-120   8  MID    Manufacturer ID    (binary)   ;\assigned by MMCA/JEDEC
  119-114   6  -      Reserved (0)                  ;
  113-112   2  CBX    Device (0=Card, 1=BGA, 2=POP) ;
  119-104   8  OID    OEM/Application ID (binary)   ;/
  103-56   48  PNM    Product name       (ASCII)
  55-48     8  PRV    Product revision   (BCD, 00h-99h) (eg 62h = rev 6.2)
  47-16    32  PSN    Product serial number (32bit)
  15-8      8  MDT    Manufacturing date (myh) (m=1..12, y=0..15; +1997)
  7-1       7  CRC    CRC7 checksum
  0         1  1      Stop bit (always 1)
```

Known CID's for DSi eMMC chips (excluding CRC in LSB, padded 00 in MSB)

```
  MY ss ss ss ss 03 4D 30 30 46 50 41 00 00 15 00  ;DSi Samsung KMAPF0000M-S998
  MY ss ss ss ss 32 57 37 31 36 35 4D 00 01 15 00  ;DSi Samsung KLM5617EFW-B301
  MY ss ss ss ss 30 36 35 32 43 4D 4D 4E 01 FE 00  ;DSi ST NAND02GAH0LZC5 rev30
  MY ss ss ss ss 31 36 35 32 43 4D 4D 4E 01 FE 00  ;DSi ST NAND02GAH0LZC5 rev31
  MY ss ss ss ss 03 47 31 30 43 4D 4D 00 01 11 00  ;3DS whatever chiptype?
  MY ss ss ss ss 07 43 59 31 47 34 4D 00 01 15 00  ;3DS Samsung KLM4G1YE0C-B301
```

See also:
DSi Console IDs

DSi SD/MMC Protocol: CSD Register (128bit Card-Specific Data)

CMD9 - SD/MMC - SPI - GET_CSD (type=ac)

Parameter bits:

```
31-16 RCA (SPI Mode: stuff bits)
15-0  stuff bits
```

SD Mode Response: R2:

```
135     Start Bit (0)                                ;\
134     Transmission To Host (0)                     ; 1st byte
133-128 Reserved (111111) (instead of Command value) ;/
127-8   CSD (120bit) (15 bytes)     ;\aka 128bit     ;-2nd..16th byte
7-1     CRC7                        ; when including ;\17th byte
0       End Bit (1)                 ;/CRC7+EndBit    ;/
```

SPI Mode Response: R1, plus DATA line,

SPI Mode Additional Data Transfer (from card):

```
127-0   CID (128bit) ... or 120bit ?
```

Addressed card sends its card-specific data (CSD).

CMD27 - SD/MMC - SPI - PROGRAM_CSD (type=adtc)

```
31-0  stuff bits
```

Response: R1

Additional Data Transfer (to card):

```
128-0  CSD register (whole 128bit) (read-only bits must be unchanged)
```

Programming of the programmable bits of the CSD, ie. the "R/W" bits, the "R" bits must be kept unchanged (read via CMD9, and write-back same values via CMD27).

The writable once "R/W(1)" bits can be changed only from 0-to-1, or vice-versa, they can be really written only ONCE, by the manufacturer?

Most of the "R/W(1)" bits are probably set by the manufacturer at time when pre-formatting the card, so they aren't actually user-writeable.

CSD Register

The types of the entries in the table below are coded as follows: R=readable, W(1)=writable once, W=multiple writable.

Bit	Siz	Type	Name	Field	Value
127-126	2	R	CSD structure version	CSD_STRUCTURE	00b
125-122	4	R	MMC: System spec version	SPEC_VERS	..
125-122	4	R	SD: reserved	-	0000b
121-120	2	R	reserved	-	00b
119-112	8	R	data read access-time-1	TAAC	xxh
111-104	8	R	data read access-time-2	NSAC	xxh
103-96	8	R	max data transfer rate	TRAN_SPEED	32h or 5Ah
95-84	12	R	card command classes	CCC	01x110110101b
83-80	4	R	max read data block len	READ_BL_LEN	xh
79	1	R	partial blocks for read allowed	READ_BL_PARTIAL	1b
78	1	R	write block misalignment	WRITE_BLK_MISALIGN	xb
77	1	R	read block misalignment	READ_BLK_MISALIGN	xb
76	1	R	DSR implemented	DSR_IMP	xb
75-74	2	R	reserved	-	00b
73-70	4	R	SDHC/SDXC: reserved	-	0000b
69-48	22	R	SDHC/SDXC: device size	C_SIZE	...
47	1	R	SDHC/SDXC: reserved	-	0
73-62	12	R	MMC/SDSC: device size	C_SIZE	xxxh
61-59	3	R	MMC/SDSC: max read current @VDD min	VDD_R_CURR_MIN	xxxb
58-56	3	R	MMC/SDSC: max read current @VDD max	VDD_R_CURR_MAX	xxxb
55-53	3	R	MMC/SDSC: max write current @VDD min	VDD_W_CURR_MIN	xxxb
52-50	3	R	MMC/SDSC: max write current @VDD max	VDD_W_CURR_MAX	xxxb
49-47	3	R	MMC/SDSC: device size multiplier	C_SIZE_MULT	xxxb
46-42	5	R	MMC: Erase Group Size	ERASE_GRP_SIZE	..
41-37	5	R	MMC: Erase Group Multiplier	ERASE_GRP_MULT	..
36-32	5	R	MMC: Write Protect Grp Size	WP_GRP_SIZE	..
46	1	R	SD: erase single block enable	ERASE_BLK_EN	xb
45-39	7	R	SD: erase sector size	SECTOR_SIZE	xxxxxxxb

```
38-32   7  R      SD: write protect group size      WP_GRP_SIZE      xxxxxxxb
31      1  R      write protect group enable        WP_GRP_ENABLE    xb
30-29   2  R      MMC: Manufacturer default ECC     DEFAULT_ECC      ..
30-29   2  R      SD: reserved (do not use)         -                00b
28-26   3  R      write speed factor                R2W_FACTOR       xxxb
25-22   4  R      max write data block len          WRITE_BL_LEN     xxxxb
21      1  R      partial blocks for write allowed  WRITE_BL_PARTIAL xb
20-17   4  R      reserved                          -                0000b
16      1  R      SD: reserved                      -                0
16      1  R      MMC: Content Protection Applicat. CONTENT_PROP_APP ..
15      1  R/W(1) File format group                 FILE_FORMAT_GRP  xb
15      1  R      SDHC/SDXC: reserved               (FILE_FORMAT_GRP)0
14      1  R/W(1) copy flag                         COPY             xb
13      1  R/W(1) permanent write protection     PERM_WRITE_PROTECT  xb
12      1  R/W    temporary write protection     TMP_WRITE_PROTECT   xb
11-10   2  R/W(1) File format                     FILE_FORMAT         xxb
11-10   2  R      SDHC/SDXC: reserved            (FILE_FORMAT)       00b
9-8     2  R/W    MMC: ECC Code                  ECC                 ..
9-8     2  R/W    SDSC: reserved, R/W            -                   00b
9-8     2  R      SDHC/SDXC: reserved, R         -                   00b
7-1     7  R/W    CRC                            CRC                 xxxxxxxb
0       1  -      not used, always '1'           -                   1b
```

Known CSD's for DSi eMMC chips (excluding CRC in LSB, padded 00 in MSB)

```
8  16 24 32 40 48 56 64 72 80 88 96 104112120pad ;<--bit numbers
40 40 96 E9 7F DB F6 DF 01 59 0F 2A 01 26 90 00  ;DSi Samsung KMAPF0000M-S998
40 40 8E FF 03 DB F6 DF 01 59 0F 32 01 27 90 00  ;DSi Samsung KLM5617EFW-B301
00 40 8A E0 BF FF 7F F5 80 59 0F 32 01 2F 90 00  ;DSi ST NAND02GAH0LZC5 rev30
00 40 8A E0 BF FF 7F F5 80 59 0F 32 01 2F 90 00  ;DSi ST NAND02GAH0LZC5 rev31
?                                             00  ;3DS whatever chiptype?
40 40 8A E7 FF DB F6 6B 02 5A 0F 32 01 27 D0 00  ;3DS Samsung KLM4G1YE0C-B301
```

That is, differences are:

```
bit     name            KMAPF0000M  KLM5617EFW NAND02GAH0LZC5 KLM4G1YE0C
126-127 CSD_STRUCTURE   2=v1.2      2=v1.2     2=v1.2         3=SeeEXT_CSD
112-119 TAAC            26h=1.5ms   27h=15ms   2Fh=20ms       27h=15ms
96-103  TRAN_SPEED      2Ah=20MHz   32h=25MHz  32h=25MHz      32h=25MHz
80-83   READ_BL_LEN     9=512       9=512      9=512          0Ah=1024
79      READ_BL_PARTIAL 0=No(?)     0=No(?)    1=Yes          0=No(?)
62-73   C_SIZE          77Fh=240MB  77Fh=240MB 3D5h=245.5MB   9AFh=1240MB
59-61   VDD_R_CURR_MIN  6=60mA      6=60mA     7=100mA        6=60mA
56-58   VDD_R_CURR_MAX  6=80mA      6=80mA     7=200mA        6=80mA
53-55   VDD_W_CURR_MIN  6=60mA      6=60mA     7=100mA        6=60mA
50-52   VDD_W_CURR_MAX  6=80mA      6=80mA     7=200mA        6=80mA
47-49   C_SIZE_MULT     6=256       6=256      7=512          7=512
42-46   ERASE_GRP_SIZE  1Fh=32x32   00h=1x32   1Fh=32x32      1Fh=32x32
32-36   WP_GRP_SIZE     09h=10      1Fh=32     00h=1          07h=8
26-28   R2W_FACTOR      05h=32x     03h=8x     02h=4x         02h=4x
14      COPY            1=Copy      1=Copy     0=Original     1=Copy
```

Not sure if that values are really correct, or if the manufacturer has screwed up some bits. TAAC being 10x slower in newer chips looks weird, 20MHz would be for 1bit MCC (whilst 4bit MMCplus/MMCmobile should support 26MHz), erase group 32x32x512 bytes would somewhat require 512Kbyte clusters, write protect group size 10 decimal looks a bit odd (though it could be true), and, well, faster writing in newer chips looks plausible.

CSD_STRUCTURE (upper 2bit of CSD register)

Field structures of the CSD register are different depend on the Physical Layer Specification Version and Card Capacity.

The CSD_STRUCTURE field in the CSD register indicates its structure version.

For MMC:

```
00h      CSD version No. 1.0    MMC Version 1.0 - 1.2
01h      CSD version No. 1.1    MMC Version 1.4 - 2.2
02h      CSD version No. 1.2    MMC Version 3.1 - 3.2 - 3.31 - 4.0 - 4.1- 4.2
03h      Version is coded in the CSD_STRUCTURE byte in the EXT_CSD register
```

For SD:

```
00h      CSD Version 1.0   SDSC (Standard Capacity)
01h      CSD Version 2.0   SDHC/SDXC (High Capacity and Extended Capacity)
02h-03h  Reserved
```

SDHC/SDXC applies major changes to CSD register (C_SIZE is expanded, and many other fields are removed or set to dummy values), for details see:
[DSi SD/MMC Protocol: CSD Register (128bit Card-Specific Data) Version 2.0]

SPEC_VERS (MMC only)

```
00h       MMC System Specification Version 1.0 - 1.2
01h       MMC System Specification Version 1.4
02h       MMC System Specification Version 2.0 - 2.2
03h       MMC System Specification Version 3.1 - 3.2 - 3.31
04h       MMC System Specification Version 4.0 - 4.1 - 4.2
05h-0Fh   Reserved
```

TAAC

Defines the asynchronous part of the data access time.

```
7    Reserved
6-3  Time value
       0=reserved, 1=1.0, 2=1.2, 3=1.3, 4=1.5, 5=2.0, 6=2.5, 7=3.0,
       8=3.5, 9=4.0, A=4.5, B=5.0, C=5.5, D=6.0, E=7.0, F=8.0
2-0  Time unit
       0=1ns, 1=10ns, 2=100ns, 3=1us, 4=10us, 5=100us, 6=1ms, 7=10ms
```

NSAC

Defines the worst case for the clock-dependent factor of the data access time. The unit for NSAC is 100 clock cycles. Therefore, the maximal value for the clock-dependent part of the data access time is 25500 clock cycles. The total access time NAC is the sum of TAAC and NSAC. It should be computed by the host for the actual clock rate. The read access time should be interpreted as a typical delay for the first data bit of a data block or stream.

TRAN_SPEED

The following table defines the maximum data transfer rate PER ONE data line:

```
7    Reserved
6-3  Time value
       0=reserved, 1=1.0, 2=1.2, 3=1.3, 4=1.5, 5=2.0, 6=2.5, 7=3.0,
       8=3.5, 9=4.0, A=4.5, B=5.0, C=5.5, D=6.0, E=7.0, F=8.0
2-0  Transfer rate unit
       0=100kbit/s, 1=1Mbit/s, 2=10Mbit/s, 3=100Mbit/s, 4..7=reserved
       MMC: same as above, but specified in <Hz> instead of <bits/s>
```

Note that for current SD Memory Cards, this field shall be always 32h which is equal to 25 MHz - the mandatory maximum operating frequency of SD Memory Card.
In High-Speed mode, this field shall be always 5Ah which is equal to 50 MHz, and when the timing mode returns to the default by CMD6 or CMD0 command, its value will be 32h.

CCC (Card Command Class)

The SD Memory Card command set is divided into subsets (command classes). A value of 1 in a CCC bit means that the corresponding command class is supported. For command class definitions, refer to Table 4-21.

```
11   Supports Command Class 11 - Function Extension Commands (SD)
10   Supports Command Class 10 - Switch Function Commands (SD)
9    Supports Command Class 9 - I/O Mode Commands (SDIO/MMCIO)
8    Supports Command Class 8 - Application-Specific Commands
7    Supports Command Class 7 - Password Lock Commands
6    Supports Command Class 6 - Block-Oriented Write Protection Commands
5    Supports Command Class 5 - Erase Commands
4    Supports Command Class 4 - Block-Oriented Write Commands
3    Supports Command Class 3 - WRITE_DAT_UNTIL_STOP (MMC)
2    Supports Command Class 2 - Block-Oriented Read Commands
```

```
  1    Supports Command Class 1 - READ_DAT_UNTIL_STOP (MMC)
  0    Supports Command Class 0 - Basic Commands
```
Same for MMC (though on MMC the classes may have different meaning?)

READ_BL_LEN
The maximum read data block length is computed as 2^READ_BL_LEN. The maximum block length might therefore be in the range 512...2048 bytes (Refer to 4.3.3 for details). Note that in an SD Memory Card the WRITE_BL_LEN is always equal to READ_BL_LEN.
```
  3-0  Setting
```
Values:
```
  00h..08h  Reserved
  09h       Block length 512 Bytes (2^9)
  0Ah       Block length 1024 Bytes (2^10)
  0Bh       Block length 2048 Bytes (2^11)
  0Ch..0Fh  Reserved
```
MMC allows any values from 2^0 to 2^14, and uses 0Fh for Extension (see TBD field in EXT_CSD) (uh, but "TBD" isn't yet defined in KMCEN0000M datasheet, maybe TBD means to-be-defined?).

READ_BL_PARTIAL (always = 1 in SDSC Memory Card)
Partial Block Read is always allowed in an SDSC Memory Card. It means that smaller blocks can be used as well. The minimum block size will be one byte.

WRITE_BLK_MISALIGN
Defines if the data block to be written by one command can be spread over more than one physical block of the memory device. The size of the memory block is defined in WRITE_BL_LEN.
```
  WRITE_BLK_MISALIGN=0   crossing physical block boundaries is invalid
  WRITE_BLK_MISALIGN=1   crossing physical block boundaries is allowed
```

READ_BLK_MISALIGN
Defines if the data block to be read by one command can be spread over more than one physical block of the memory device. The size of the memory block is defined in READ_BL_LEN.
```
  READ_BLK_MISALIGN=0    crossing physical block boundaries is invalid
  READ_BLK_MISALIGN=1    crossing physical block boundaries is allowed
```

DSR_IMP
Defines if the configurable driver stage is integrated on the card. If set, a driver stage register (DSR) shall be implemented (also see Chapter 5.5).
```
  DSR_IMP=0    no DSR implemented
  DSR_IMP=1    DSR implemented
```

C_SIZE (for max 2GB)
This parameter is used to compute the user's data card capacity (not include the security protected area). The memory capacity of the card is computed from the entries C_SIZE, C_SIZE_MULT and READ_BL_LEN as follows:
```
  memory capacity = BLOCKNR * BLOCK_LEN
```
Whereas,
```
  BLOCKNR = (C_SIZE+1) * MULT
  MULT = 2^(C_SIZE_MULT+2)    ;(C_SIZE_MULT < 8)
  BLOCK_LEN = 2^READ_BL_LEN   ;(READ_BL_LEN < 12)
```
To indicate 2 GByte card, BLOCK_LEN shall be 1024 bytes.
Therefore, the maximal capacity that can be coded is 4096*512*1024 = 2 G bytes.
Example: A 32 Mbyte card with BLOCK_LEN = 512 can be coded by C_SIZE_MULT = 3 and C_SIZE = 2000.
The Maximum Data Area size of SDSC Standard Capacity Card is 4,153,344 sectors (2028MB).

C_SIZE (for MMC above 2GB)
The 12bit C_SIZE field should be set 0FFFh. Use "SEC_COUNT" in EXT_CSD to specify actual size.

VDD_R_CURR_MIN, VDD_W_CURR_MIN
Maximum values for read and write currents at the MINIMAL power supply VDD:

```
2-0   0=0.5mA, 1=1mA, 2=5mA, 3=10mA, 4=25mA, 5=35mA, 6=60mA, 7=100mA
```

VDD_R_CURR_MAX, VDD_W_CURR_MAX
Maximum values for read and write currents at the MAXIMAL power supply VDD:

```
2-0   0=1mA, 1=5mA, 2=10mA, 3=25mA, 4=35mA, 5=45mA, 6=80mA, 7=200mA
```

C_SIZE_MULT
This parameter is used for coding a factor MULT for computing the total device size (see 'C_SIZE'). Defined as "MULT = 2^(C_SIZE_MULT+2)".

```
2-0   Device Size Factor (0..7 = Factor 4,8,16,32,64,128,256,512)
```

SD: ERASE_BLK_EN
The ERASE_BLK_EN defines the granularity of the unit size of the data to be erased. The erase operation can erase either one or multiple units of 512 bytes or one or multiple units (or sectors) of SECTOR_SIZE (see definition below).
If ERASE_BLK_EN=0, the host can erase one or multiple units of SECTOR_SIZE. The erase will start from the beginning of the sector that contains the start address to the end of the sector that contains the end address. For example, if SECTOR_SIZE=31 and the host sets the Erase Start Address to 5 and the Erase End Address to 40, the physical blocks from 0 to 63 will be erased as shown in Figure 5-1.

```
Figure 5-1: ERASE_BLK_EN = 0 Example
Physical Block (per CSD)
0          1          2          3          4          5          6
0123456789 0123456789 0123456789 0123456789 0123456789 0123456789 0123456789
     <----- Host Erase Address Range ------->
<---------- Erase Area ---------------------------------------------->
<---------- Erase Unit Size ------><------- Erase Unit Size --------->
```

If ERASE_BLK_EN=1 the host can erase one or multiple units of 512 bytes. All blocks that contain data from start address to end address are erased. For example, if the host sets the Erase Start Address to 5 and the Erase End Address to 40, the physical blocks from 5 to 40 will be erased as shown in Figure 5-2.

```
Figure 5-2: ERASE_BLK_EN = 1 Example
Physical Block (per CSD)
0          1          2          3          4          5          6
0123456789 0123456789 0123456789 0123456789 0123456789 0123456789 0123456789
     <----- Host Erase Address Range ------->
     <----- Erase Area ---------------------->
```

SD: SECTOR_SIZE
The size of an erasable sector. The content of this register is a 7-bit binary coded value, defining the number of write blocks (see WRITE_BL_LEN). The actual size is computed by increasing this number by one. A value of zero means one write block, 127 means 128 write blocks.

MMC: ERASE_GRP_SIZE
The contents of this register is a 5 bit binary coded value, used to calculate the size of the erasable unit of the moviNAND. The size of the erase unit (also referred to as erase group) is determined by the ERASE_GRP_SIZE and the ERASE_GRP_MULT entries of the CSD, using the following equation:

```
size of erasable unit = (ERASE_GRP_SIZE + 1) * (ERASE_GRP_MULT + 1)
```

This size is given as minimum number of write blocks that can be erased in a single erase command.

MMC: ERASE_GRP_MULT
A 5 bit binary coded value used for calculating the size of the erasable unit of the moviNAND. See ERASE_GRP_SIZE section for detailed description.

MMC: DEFAULT_ECC
Set by the moviNAND manufacturer. It defines the ECC code which is recommended for use. The field definition is the same as for the ECC field described later.

MMC: CONTENT_PROT_APP
This field in the CSD indicates whether the content protection application is supported. MultiMediaCards which implement the content protection application will have this bit set to '1'.

MMC: ECC
Defines the ECC code that was used for storing data on the moviNAND. This field is used by the host (or application) to decode the user data. The following table defines the field format:

ECC	ECC type	Maximum number of correctable bits per block
00h	None (default)	Mone
01h	BCH (542,512)	3
02h-03h	Reserved	-

MMC: WP_GRP_SIZE (5bit)
The size of a write protected group. The contents of this register is a 5 bit binary coded value, defining the number of erase groups which can be write protected.
The actual size is computed by increasing this number by one. A value of zero means 1 erase group, 31 means 32 erase groups. (Refer to the chapter 4.11.1 on page 48)

SD: WP_GRP_SIZE (7bit)
The size of a write protected group. The content of this register is a 7-bit binary coded value, defining the number of erase sectors (see SECTOR_SIZE). The actual size is computed by increasing this number by one. A value of zero means one erase sector, 127 means 128 erase sectors.

WP_GRP_ENABLE
A value of 0 means no group write protection possible.

R2W_FACTOR
Defines the typical block program time as a multiple of the read access time.

2-0	Multiples of read access time (0..5=Mul 1,2,4,8,16,32, 6..7=Reserved)

For example, value 5 means that writing is 32 times slower than reading.

WRITE_BL_LEN
The maximum write data block length is computed as 2^WRITE_BL_LEN. The maximum block length might therefore be in the range from 512 to 2048 bytes. Write Block Length of 512 bytes is always supported.
Note that in the SD Memory Card, the WRITE_BL_LEN is always equal to READ_BL_LEN.

3-0	Block Length

Values:

00h..08h	Reserved
09h	512 bytes (2^9)
0Ah	1024 Bytes (2^10)
0Bh	2048 Bytes (2^11)
0Ch..0Fh	Reserved

MMC: See READ_BL_LEN

WRITE_BL_PARTIAL
Defines whether partial block sizes can be used in block write commands.
WRITE_BL_PARTIAL=0 means that only the WRITE_BL_LEN block size and its partial derivatives, in resolution of units of 512 bytes, can be used for block oriented data write.
WRITE_BL_PARTIAL=1 means that smaller blocks can be used as well. The minimum block size is one byte.

COPY
Defines whether the contents is original (=0) or has been copied (=1). Setting this bit to 1 indicates that the card content is a copy. The COPY bit is a one time programmable bit except ROM card.

PERM_WRITE_PROTECT

TMP_WRITE_PROTECT
Permanently/temporarily write-protects the entire card (by disabling all write and erase commands). The default values are 0, ie. not write protected.

FILE_FORMAT
FILE_FORMAT_GRP
Indicates the file format on the card. These fields are read-only for ROM. The following formats are defined:

FILE_FORMAT_GRP	FILE_FORMAT	Type
0	0	Hard disk-like file system with partition table
0	1	DOS FAT (floppy-like) with boot sector only (no partition table)
0	2	Universal File Format
0	3	Others/Unknown
1	0, 1, 2, 3	Reserved

A more detailed description is given in the Filesystem Specification.

DSi SD/MMC Protocol: CSD Register (128bit Card-Specific Data) Version 2.0

CSD Register (CSD Version 2.0) (SDHC/SDXC)
The field name in parenthesis is set to fixed value and indicates that the host is not necessary to refer these fields. The fixed values enables host, which refers to these fields, to keep compatibility to CSD Version 1.0. The Cell Type field is coded as follows: R=readable, W(1)=writable once, W=multiple writable.

Bit	Siz	Type	Name	Field	Value
127-126	2	R	CSD structure	CSD_STRUCTURE	01b
125-120	6	R	reserved	-	000000b
119-112	8	R	data read access-time-1	(TAAC)	0Eh
111-104	8	R	data read access-time-2	(NSAC)	00h
103-96	8	R	max data transfer rate	(TRAN_SPEED)	32h,5Ah,0Bh,2Bh
95-84	12	R	card command classes	CCC	x1x110110101b
83-80	4	R	max read data block length	(READ_BL_LEN)	9
79	1	R	partial blocks for read allowed	(READ_BL_PARTIAL)	0
78	1	R	write block misalignment	(WRITE_BLK_MISALIGN)	0
77	1	R	read block misalignment	(READ_BLK_MISALIGN)	0
76	1	R	DSR implemented	DSR_IMP	x
75-70	6	R	reserved	-	000000b
69-48	22	R	device size	C_SIZE	xxxxxxh
47	1	R	reserved	-	0
46	1	R	erase single block enable	(ERASE_BLK_EN)	1
45-39	7	R	erase sector size	(SECTOR_SIZE)	7Fh
38-32	7	R	write protect group size	(WP_GRP_SIZE)	00h
31	1	R	write protect group enable	(WP_GRP_ENABLE)	0
30-29	2	R	reserved	-	00b
28-26	3	R	write speed factor	(R2W_FACTOR)	010b
25-22	4	R	max write data block length	(WRITE_BL_LEN)	9
21	1	R	partial blocks for write allowed	(WRITE_BL_PARTIAL)	0
20-16	5	R	reserved	-	00000b
15	1	R	File format group	(FILE_FORMAT_GRP)	0
14	1	R/W(1)	copy flag	COPY	x
13	1	R/W(1)	permanent write protection	PERM_WRITE_PROTECT	x
12	1	R/W	temporary write protection	TMP_WRITE_PROTECT	x
11-10	2	R	File format	(FILE_FORMAT)	00b
9-8	2	R	reserved	-	00b
7-1	7	R/W	CRC	CRC	xxh
0	1	-	not used, always '1'	-	1

C_SIZE
This field is expanded to 22 bits and can indicate up to 2 TBytes (that is the same as the maximum memory

space specified by a 32-bit block address.)
This parameter is used to calculate the user data area capacity in the SD memory card (not include the protected area). The user data area capacity is calculated from C_SIZE as follows:

```
memory capacity = (C_SIZE+1) * 512KByte
```

The Minimum user area size of SDHC Card is 4,211,712 sectors (2GB + 8.5MB).
The Minimum value of C_SIZE for SDHC in CSD Version 2.0 is 001010h (4112).
The Maximum user area size of SDHC Card is (32GB - 80MB).
The Maximum value of C_SIZE for SDHC in CSD Version 2.0 is 00FF5Fh (65375).
The Minimum user area size of SDXC Card is 67,108,864 sectors (32GB).
The Minimum value of C_SIZE for SDXC in CSD Version 2.0 is 00FFFFh (65535).

TRAN_SPEED
TRAN_SPEED is variable depends on bus speed mode of SD Interface.
When CMD0 is received, this field is reset to 32h.
On SDSC (but not SDHC/SDXC), CMD6 does the same reset stuff?

```
32h   SDSC/SDHC/SDXC in Default Speed mode (25MHz)
5Ah   SDSC/SDHC/SDXC in High Speed mode    (50MHz)
0Bh   SDHC/SDXC in SDR50 or DDR50 mode     (100Mbit/sec)
2Bh   SDHC/SDXC in SDR104 mode             (200Mbit/sec)
```

UHS-II mode is not related to this field.

CCC, DSR_IMP, COPY, PERM_WRITE_PROTECT, TMP_WRITE_PROTECT
Definition of these fields is same as in CSD Version 1.0.

TAAC, NSAC, R2W_FACTOR
In SDHC/SDXC, these fields should be fixed to TAAC=0Eh (1 ms), NSAC=00h, and R2W_FACTOR=02h (mul4).
The host should not use TAAC, NSAC, and R2W_FACTOR to calculate timeout and should uses fixed timeout values for read and write operations (See 4.6.2).

READ_BL_LEN, WRITE_BL_LEN
These two fields are fixed to 9h (which indicates 512 Bytes).

READ_BL_PARTIAL, WRITE_BL_PARTIAL, READ_BLK_MISALIGN, WRITE_BLK_MISALIGN
These four fields are fixed to 0 (partial block read and physical page crossing prohibited for block read/write).

SECTOR_SIZE
This field is fixed to 7Fh, which indicates 64 KBytes. This value is not related to erase operation. SDHC and SDXC Cards indicate memory boundary by AU size and this field should not be used.

ERASE_BLK_EN
This field is fixed to 1, which means the host can erase one or multiple units of 512 bytes.

WP_GRP_SIZE, WP_GRP_ENABLE
These field are fixed to WP_GRP_SIZE=00h, and WP_GRP_ENABLE=0.
SDHC and SDXC Cards do not support write protected groups.

FILE_FORMAT_GRP
FILE_FORMAT
These fields are set to 0. Host should not use these fields.

DSi SD/MMC Protocol: EXT_CSD Register (4096bit Extended CSD Register) (MMC)

CMD8 - MMC - SPI - GET_EXT_CSD (type=adtc)

```
  31-0  stuff bits
```

Response: R1

Additional Data Transfer (from card):

```
  4095-0  EXT_CSD Register (4096bit)
```

MMC only.

CMD6 - MMC - SPI - SWITCH (type=ac)

```
  31-26  6bit  Reserved (0)
  25-24  2bit  Access
                 00h  Change Command Set (EXT_CSD[191] = parameter bit2-0)
                 01h  Set bits (EXT_CSD[index] = EXT_CSD[index] OR value)
                 02h  Clr bits (EXT_CSD[index] = EXT_CSD[index] AND NOT value)
                 03h  Write    (EXT_CSD[index] = value)
  23-16  8bit  Index (0..191)   ;\used only if "Access=1..3"
  15-8   8bit  Value (0..255)   ;/
  7-3    5bit  Reserved (0)
  2-0    3bit  Cmd Set (0..7)   ;-used only if "Access=0"
```

Response: R1b

Additional Data Transfer (from card): Busy signal for "R1b" response

MMC only.

Extended CSD Register (MMC only)

The Extended CSD register defines the card properties and selected modes. It is 512 bytes (4096 bits) long. The most significant 320 bytes are the Properties segment, which defines the card capabilities and cannot be modified by the host. The lower 192 bytes are the Modes segment, which defines the configuration the card is working in. These modes can be changed by the host by means of the SWITCH command.

Byte	Siz	Type	Name	Field
Properties Segment				
511-505	7	-	Reserved(1)	-
504	1	R	Supported Command Sets	S_CMD_SET
503-216	288	-	Reserved(1)	-
215-212	4	R	moviNAND only: Sector Count	SEC_COUNT
211	1	-	Reserved	-
210	1	R	Min Write Performance for 8bit @52MHz	MIN_PERF_W_8_52
209	1	R	Min Read Performance for 8bit @52MHz	MIN_PERF_R_8_52
208	1	R	Min Write Perf. for 8/4bit @26/52MHz	MIN_PERF_W_8_26_4_52
207	1	R	Min Read Perf. for 8/4bit @26/52MHz	MIN_PERF_R_8_26_4_52
206	1	R	Min Write Performance for 4bit @26MHz	MIN_PERF_W_4_26
205	1	R	Min Read Performance for 4bit @26MHz	MIN_PERF_R_4_26
204	1	-	Reserved(1)	-
203	1	R	Power Class for 26MHz @ 3.6V	PWR_CL_26_360
202	1	R	Power Class for 52MHz @ 3.6V	PWR_CL_52_360
201	1	R	Power Class for 26MHz @ 1.95V	PWR_CL_26_195
200	1	R	Power Class for 52MHz @ 1.95V	PWR_CL_52_195
199-197	3	-	Reserved(1)	-
196	1	R	Card Type	CARD_TYPE
195	1	-	Reserved(1)	-
194	1	R	CSD Structure Version	CSD_STRUCTURE
193	1	-	Reserved(1)	-
192 C0h	1	R	Extended CSD Revision	EXT_CSD_REV
Modes Segment				
191 BFh	1	R/W	Command Set	CMD_SET
190 BEh	1	-	Reserved(1)	-
189 BDh	1	RO	Command Set Revision	CMD_SET_REV
188 BCh	1	-	Reserved(1)	-
187 BBh	1	R/W	Power Class	POWER_CLASS
186 BAh	1	-	Reserved(1)	-
185 B9h	1	R/W	High Speed Interface Timing	HS_TIMING
184 B8h	1	-	Reserved(1)	-
183 B7h	1	WO	Bus Width Mode	BUS_WIDTH

```
  182 B6h      ?     ?
  181 B5h   1  -     Reserved                                -
  180 B4h   1  RO    moviNAND only: Erased Memory Content  ERASED_MEM_CONT
  180-0   181  -     Reserved(a)                             -
```
(a) Reserved(a) bits should read as '0'.
(1) Reserved(1) bits should be probably ZERO, too.
The above table is transferred "most significant first", which does probably mean that it starts with BYTE 0, not with byte 511. ALTHOUGH, the 4-byte SEC_COUNT appears to be LITTLE-ENDIAN?
Note: JEDEC Standard No. 84-A44 contains MANY additional fields in EXT_CSD.

S_CMD_SET
This field defines which command sets are supported by the card.

```
  Bit   Command Set
  7-5   Reserved
  4     moviNAND only: ATA on MMC
  3     moviNAND only: SecureMCC 2.0
  2     Content Protection SecureMMC
  1     SecureMMC
  0     Standard MMC
```

SEC_COUNT (moviNAND and newer JEDEC specs only)
The device density is calculated from the register by multiplying the value of the register (sector count) by 512B/sector. The maximum density possible to be indicated is thus 2 Tera bytes (minus 512 bytes) (4,294,967,295 x 512B). The least significant byte (LSB) of the sector count value is the byte [212].

MIN_PERF_a_b_ff
These fields defines the overall minimum performance value for the read and write access with different bus width and max clock frequency modes. The value in the register is coded as follows. Other than defined values are illegal.

```
  Value Performance
  0x00  For Cards not reaching the 2.4MB/s minimum value
  0x08  Class A: 2.4MB/s and is the lowest allowed value for MMCplus and
         MMCmobile(16x150kB/s)
  0x0A  Class B: 3.0MB/s and is the next allowed value (20x150kB/s)
  0x0F  Class C: 4.5MB/s and is the next allowed value (30x150kB/s)
  0x14  Class D: 6.0MB/s and is the next allowed value (40x150kB/s)
  0x1E  Class E: 9.0MB/s and is the next allowed value (60x150kB/s)
         This is also the highest class which any MMCplus or MMCmobile card
         is needed to support in low bus category operation mode (26MHz with
         4bit data bus).
         A MMCplus or MMCmobile card supporting any higher class than this
         have to support this class also (in low category bus operation mode).
  0x28  Class F: Equals 12.0MB/s and is the next allowed value (80x150kB/s)
  0x32  Class G: Equals 15.0MB/s and is the next allowed value (100x150kB/s)
  0x3C  Class H: Equals 18.0MB/s and is the next allowed value (120x150kB/s)
  0x46  Class J: Equals 21.0MB/s and is the next allowed value (140x150kB/s)
         This is also the highest class which any MMCplus or MMCmobile card
         is needed to support in mid bus category operation mode (26MHz with
         8bit data bus or 52MHz with 4bit data bus).
         A MMCplus or MMCmobile card supporting any higher class than this
         have to support this Class (in mid category bus operation mode) and
         Class E also (in low category bus operation mode).
  0x50  Class K: Equals 24.0MB/s and is the next allowed value (160x150kB/s)
  0x64  Class M: Equals 30.0MB/s and is the next allowed value (200x150kB/s)
  0x78  Class O: Equals 36.0MB/s and is the next allowed value (240x150kB/s)
  0x8C  Class R: Equals 42.0MB/s and is the next allowed value (280x150kB/s)
  0xA0  Class T: Equals 48.0MB/s and is the last defined value (320x150kB/s)
```

PWR_CL_ff_vvv
These fields define the supported power classes by the card. By default, the card has to operate at maximum

frequency using 1 bit bus configuration, within the default max current consumption, as stated in the table below. If 4 bit/8 bits bus configurations, require increased current consumption, it has to be stated in these registers.
By reading these registers the host can determine the power consumption of the card in different bus modes. Bits [7:4] code the current consumption for the 8 bit bus configuration. Bits [3:0] code the current consumption for the 4 bit bus configuration.
The PWR_52_vvv registers are not defined for 26MHz MultiMediaCards.

Voltage	Value	Max RMS Current	Max Peak Current	Remarks
3.6V	0	100 mA	200 mA	Default current
	1	120 mA	220 mA	consumption for
	2	150 mA	250 mA	high voltage cards
	3	180 mA	280 mA	
	4	200 mA	300 mA	
	5	220 mA	320 mA	
	6	250 mA	350 mA	
	7	300 mA	400 mA	
	8	350 mA	450 mA	
	9	400 mA	500 mA	
	10	450 mA	550 mA	
	11-15	Reserved for future use		
1.95V	0	65 mA	130 mA	Default current
	1	70 mA	140 mA	consumption for
	2	80 mA	160 mA	Dual voltage cards
	3	90 mA	180 mA	(if any, not moviNAND)
	4	100 mA	200 mA	
	5	120 mA	220 mA	
	6	140 mA	240 mA	
	7	160 mA	260 mA	
	8	180 mA	280 mA	
	9	200 mA	300 mA	
	10	250 mA	350 mA	
	6-15	Reserved for future use		

The measurement for max RMS current is done as average RMS current consumption over a period of 100ms. Max peak current is defined as absolute max value not to be exceeded at all.
The conditions under which the power classes are defined are:

- Maximum bus frequency
- Maximum operating voltage
- Worst case functional operation
- Worst case environmental parameters (temperature,...)

These registers define the maximum power consumption for any protocol operation in data transfer mode, Ready state and Identification state.

CARD_TYPE
This field defines the type of the card. The only currently valid values for this field are 0x01 and 0x03.

Bit	Card Type
7-2	Reserved
1	High Speed MultiMediaCard @ 52MHz
0	High Speed MultiMediaCard @ 26MHz

CSD_STRUCTURE
This field is a continuation of the CSD_STRUCTURE field in the CSD register.

CSD_STRUCTURE	CSD structure version	Valid for System Specification Version
0	CSD version No. 1.0	Version 1.0 - 1.2
1	CSD version No. 1.1	Version 1.4 - 2.2
2	CSD version No. 1.2	Version 3.1-3.2-3.31-4.0-4.1-4.2
3	Reserved for future use	
4-255	Reserved for future use	

EXT_CSD_REV
Defines the fixed parameters. related to the EXT_CSD, according to its revision.

EXT_CSD_REV	Extended CSD Revision

```
0             Revision 1.0
1             Revision 1.1
2             Revision 1.2 (moviNAND)
3-255         Reserved
```

CMD_SET
Contains the binary code of the command set that is currently active in the card. It is set to '0' (Standard MMC) after power up and can be changed by a SWITCH command.

CMD_SET_REV
Contains a binary number reflecting the revision of the currently active command set. For Standard MMC command set it is:

```
Code    MMC Revisions
0       v4.0
1-255   Reserved
```

This field, though in the Modes segment of the EXT_CSD, is read only.

POWER_CLASS
This field contains the 4 bit value of the selected power class for the card. The power classes are defined in Table. The host should be responsible of properly writing this field with the maximum power class it allows the card to use. The card uses this information to, internally, manage the power budget and deliver an optimized performance.
This field is 0 after power-on or software reset.

```
Bits    Description
7-4     Reserved
3-0     Card power class code (See Table 5-29)
```

HS_TIMING
This field is 0 after power-on, or software reset, thus selecting the backwards compatibility interface timing for the card. If the host writes 1 to this field, the card changes its timing to high speed interface timing (refer to Chapter 5.4.8).

BUS_WIDTH
It is set to '0' (1 bit data bus) after power up and can be changed by a SWITCH command.

```
Value   Bus Mode
0       1 bit data bus (MMC, with old 7pin connector)
1       4 bit data bus (MMCplus, with SD-card-compatible 9pin connector)
2       8 bit data bus (MMCplus, with special 13pin connector)
3-255   Reserved
```

For detecting cards with 4bit/8bit data bus support: Switch the SD/MMC controller to 4bit/8bit modes, and use BUSTEST_W and BUSTEST_R to test if the card sends a proper response, see https://www.mikrocontroller.net/attachment/101561/AN_MMCA050419.pdf
Note: The SD/MMC controller in the DSi supports 1bit/4bit modes only (no 8bit mode). For the DSi's onboard eMMC it's safe to assume 4bit being supported, however, external MMC cards do require detecting 4bit support.

ERASED_MEM_CONT (moviNAND) (but, RESERVED in newer JEDEC specs!)
This Field defines the content of an explicitly erased memory range.

```
Value   Erased Memory content
00h     Erased memory range shall be '0'
01h     Erased memory range shall be '1'
02h-FFh Reserved
```

Looks like a mis-definition, since value 00h should have been kept reserved for cards that do not specify whether they set erased bits to 0 or 1.

DSi SD/MMC Protocol: RCA Register (16bit Relative Card Address)

The RCA was intended for connecting multiple cards to the same host, possibly even sharing the same signal wires for multiple cards. The multi-card feature isn't used to often though.
Most hosts are having only a single card slot. And, hosts that <do> support multiple cards may use separate busses and even separate controllers for each card (eg. Nintendo DSi is doing so for onboard NAND and external SD slot).
However, even single-card systems will need to obtain a "dummy" RCA, and use that RCA value for selecting the card.
The only exception is SPI mode: SPI isn't using RCA, and doesn't support RCA commands at all - instead, in SPI mode, the cards are selected via /CS signal (which may include multiple /CS signals for multiple cards).

CMD3 - SD - GET_RELATIVE_ADDR (type=bcr)
Parameter bits:
```
  31-0  stuff bits
```
Response: R6:
```
  47      Start Bit (0)                                ;\
  46      Transmission To Host (0)                     ; 1st byte
  45-40   Command (the 6bit CMD being responded to)    ;/
  39-24   New published RCA of the card   ;-16bit      ;-2nd..3th byte
  23-22   CSR Card Status, bit 23-22      ;\           ;\
  21      CSR Card Status, bit 19         ; 16bit      ; 4nd..5th byte
  20-8    CSR Card Status, bit 12-0       ;/           ;/
  7-1     CRC7                                         ;\6th byte
  0       End Bit (1)                                  ;/
```
Ask the card to publish a new relative address (RCA).
Dunno how this is intended to work with multiple cards. The goal should be to assign <different> RCAs to each card. The command should be probably repeatedly used until all cards respond with different RCAs. This would require the cards to contain some sort of analog random generator - or maybe to use the CID register as random seed (the CID seems to contain unique serial numbers per card)?

CMD3 - MMC - SET_RELATIVE_ADDR (type=ac)
Parameter bits:
```
  31-16 RCA
  15-0  stuff bits
```
Response: R1
Assigns an RCA value TO the card (ie. the opposite of CMD3 on SD Cards).
Dunno how this is intended to work with multiple cards. The goal should be to assign <different> RCAs to each card. But actually, the command appears to assign the <same> RCA to all cards?

CMD7 - SD/MMC - SELECT_DESELECT_CARD (type=ac) ;actually: (type=bcr)
Parameter bits:
```
  31-16 RCA
  15-0  stuff bits
```
Response: R1b (only from the selected card)
Additional Data Transfer (from card): Busy signal for "R1b" response
Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all.
In the case that the RCA equals 0, then the host may do one of the following:
- Use other RCA number to perform card de-selection.
- Re-send CMD3 to change its RCA number to other than 0 and then use CMD7 with RCA=0 for card deselection.

CMD5 - MMC - SLEEP_AWAKE (type=ac)
Parameter bits:
```
  31-16 RCA
  15    Sleep/Awake flag (0=Awake/stby, 1=Sleep/slp)
```

```
  14-0  stuff bits
```
Response: R1b

RCA register
The writable 16-bit relative card address register carries the card address that is published by the card during the card identification. This address is used for the addressed host-card communication after the card identification procedure. The default value of the RCA register is 0000h. The value 0000h is reserved to set all cards into the Stand-by State with CMD7.
In UHS-II mode, Node ID is used as RCA. Refer to SD-TRAN Section of UHS-II Addendum for more details.

Note
Commands GET_CSD, GET_CID, APP_CMD, GO_INACTIVE_STATE, and GET_STATUS allow/require to specify RCA in parameter field.
Other commands are either processed by all cards (broadcast commands), or processed only be cards that have been previously selected via CMD7 (most normal commands).

Broadcast Commands
```
  CMD0     sd/mmc spi GO_IDLE_STATE (type=bc)
  CMD2     sd/mmc     ALL_GET_CID (type=bcr)
  CMD3     sd         GET_RELATIVE_ADDR (type=bcr)
  CMD4     sd/mmc     SET_DSR (type=bc)
  CMD7     sd/mmc     SELECT_DESELECT_CARD (type=ac) ;actually: (type=bcr)
  CMD8     sd     spi SET_IF_COND (type=bcr)
  ACMD41   sd     spi SD_SEND_OP_COND (type=bcr)  ;SPI: reduced functionality
```
Some broadcast commands are sending responses.
SD specs are suggesting to use separate CMD lines for each card (so the host would broadcast the same command on all CMD lines, and would receive separate responses in parallel from each CMD line).
MMC cards are said to support open-collector CMD lines (so responses from separate cards would be logically ORed, though, dunno what that would be good for).

DSi SD/MMC Protocol: DSR Register (16bit Driver Stage Register) (Optional)

CMD4 - SD/MMC - SET_DSR (type=bc)
Parameter bits:
```
  31-16 DSR
  15-0  stuff bits
```
Response: N/A
Programs the DSR of all cards.

DSR register (Optional)
The 16-bit driver stage register is described in detail in Chapter 0 (uh, where?). It can be optionally used to improve the bus performance for extended operating conditions (depending on parameters like bus length, transfer rate or number of cards). The CSD register carries the information about the DSR register usage. The default value of the DSR register is 0404h.

DSi SD/MMC Protocol: SCR Register (64bit SD Card Configuration Register)

ACMD51 - SD - SPI - GET_SCR (type=adtc)
```
  31-0  stuff bits
```

Response: R1
Additional Data Transfer (from card):

```
63-0  SCR Register (8bytes, aka 64bit)
```

SD Configuration Register (SCR)

In addition to the CSD register, there is another configuration register named SD CARD Configuration Register (SCR). SCR provides information on the SD Memory Card's special features that were configured into the given card. The size of SCR register is 64 bits. This register shall be set in the factory by the SD Memory Card manufacturer.

```
Bit  Siz  Typ Description                    Field                ;common
63-60  4  R   SCR Structure                  SCR_STRUCTURE        ;\00h or
59-56  4  R   SD Memory Card - Spec. Version SD_SPEC              ;/01h
55     1  R   data_status_after erases       DATA_STAT_AFTER_ERASE ;\
54-52  3  R   CPRM Security Support          SD_SECURITY          ; A5h
51-48  4  R   DAT Bus widths supported       SD_BUS_WIDTHS        ;/
47     1  R   Spec. Version 3.00 or higher   SD_SPEC3             ;\
46-43  4  R   Extended Security Support      EX_SECURITY          ; 0000h
42     1  R   Spec. Version 4.00 or higher   SD_SPEC4             ;
41-36  6  R   Reserved                       -                    ;
35-32  4  R   Command Support bits           CMD_SUPPORT          ;/
31-0  32  R   reserved for manufacturer usage -                   ;-var
```

SCR Register Structure Version

```
SCR_STRUCTURE  SCR Structure Version  SD Physical Layer Specification Version
00h            SCR version 1.0        Version 1.01-4.00
01h..0Fh       reserved
```

Note: SD_SPEC is used to indicate SCR Structure Version instead of this field.

SD_SPEC, SD_SPEC3, SD_SPEC4

The Physical Layer Specification Version is indicated in combination with SD_SPEC, SD_SPEC3 and SD_SPEC4 as described Table 5-19.

```
SD_SPEC  SD_SPEC3  SD_SPEC4   Physical Layer Specification Version Number
0        0         0          Version 1.0 and 1.01
1        0         0          Version 1.10
2        0         0          Version 2.00
2        1         0          Version 3.0X
2        1         1          Version 4.XX
       Others                 Reserved
```

(1) Version 2.00 hosts do not recognize SD_SPEC3 and SD_SPEC4.
(2) Version 3.00 hosts do not recognize SD_SPEC4.
Hosts recognize Physical Layer Specification Version shall also recognize including future version. Next version will be defined in SD_SPEC field.

The card manufacturer determines SD_SPEC value by conditions indicated below. All conditions shall be satisfied for each version. The other combination of conditions is not allowed.

Essential conditions to indicate Version 1.01 Card (SD_SPEC=0, SD_SPEC3=0 and SD_SPEC4=0):

```
(1) The card does not support CMD6
(2) The card does not support CMD8
(3) User area capacity shall be up to 2GB
```

Essential conditions to indicate Version 1.10 Card (SD_SPEC=1, SD_SPEC3=0 and SD_SPEC4=0):

```
(1) The card shall support CMD6
(2) The card does not support CMD8
(3) User area capacity shall be up to 2GB
```

Essential conditions to indicate Version 2.00 Card (SD_SPEC=2, SD_SPEC3=0 and SD_SPEC4=0):

```
(1) The card shall support CMD6
(2) The card shall support CMD8
(3) The card shall support CMD42
(4) User area capacity shall be up to 2GB (SDSC) or 32GB (SDHC)
(5) Speed Class shall be supported (SDHC)
```

Essential conditions to indicate Version 3.00 Card (SD_SPEC=2, SD_SPEC3=1 and SD_SPEC4=0):

```
(1) The card shall support CMD6
(2) The card shall support CMD8
(3) The card shall support CMD42
(4) User area capacity shall be up to 2GB (SDSC) or 32GB (SDHC)
  User area capacity shall be more than or equal to 32GB and up to 2TB (SDXC)
(5) Speed Class shall be supported (SDHC or SDXC)
```

Optional conditions to indicate Version 3.00 Card:

```
A card supports any of following functions shall satisfy essential
conditions of Version 3.00 Card
(1) Speed Class supported under the conditions defined in Version 3.00
(2) UHS-I supported card
(3) CMD23 supported card
```

Essential conditions to indicate Version 4.XX Card (SD_SPEC=2, SD_SPEC3=1 and SD_SPEC4=1):

```
(1) Same as the essential conditions of Version 3.00 device
(2) Support any of additional functions defined by Version 4.XX:
    Followings functions (a) to (c) are defined by Version 4.00.
      (a) Support of CMD48 and CMD49
      (b) Support of UHS-II mode
      (c) Support of DPS (Data Protection System)
    Followings functions (d) to (f) are defined by Version 4.10.
      (d) Support of CMD58 and CMD59
      (e) Support of Power Management Functions
      (f) Support of Speed Grade 1 for UHS-II mode
```

The requirements of supporting commands mentioned above are for the optional commands, the support of which depends on versions (SD_SPEC, SD_SPEC3 and SD_SPEC4). Refer to Table 4-21 (and Notes below the table) about the mandatory and optional commands in the card.

DATA_STAT_AFTER_ERASE

Defines the data status after erase, whether it is 0 or 1 (the status is card vendor dependent).

SD_SECURITY

This field indicates CPRM Security Specification Version for each capacity card. The definition of Protected Area is different in each capacity card.

```
00h          No Security
01h          Not Used
02h          SDSC Card (CPRM Security Version 1.01)
03h          SDHC Card (CPRM Security Version 2.00)
04h          SDXC Card (CPRM Security Version 3.xx)
05h-07h      Reserved
```

The basic rule of setting this field:

```
SDSC Card sets this field to 2 (Version 1.01).
SDHC Card sets this field to 3 (Version 2.00).
SDXC Card sets this field to 4 (Version 3.xx).
```

Note that it is mandatory for a regular writable SD Memory Card to support Security Protocol. For ROM (Read Only) and OTP (One Time Programmable) types of the SD Memory Card, the security feature is optional.

SD_BUS_WIDTHS

Describes all the DAT bus widths that are supported by this card.

```
Bit 3         Reserved
Bit 2         4 bit (DAT0-3)
Bit 1         Reserved
Bit 0         1 bit (DAT0)
```

Since the SD Memory Card shall support at least the two bus modes 1-bit or 4-bit width, then any SD Card shall set at least bits 0 and 2 (SD_BUS_WIDTH="0101").

EX_SECURITY

This field indicates Extended Security which is defined by the Part A4 Data Protection System Specification Version 1.00 or will be defined by a later version of the Part 3 Security Specification Version 3.00.

```
00h         Extended Security is not supported.
```

```
  01h..0Fh     Extended Security is supported. SCR[44-43] is defined by the
                 Part A4 Data Protection System Specification. SCR[46-45] is
                 reserved for future extension.
```

CMD_SUPPORT

Support bit of new commands are defined to Bit 33-32 (uh, 35-32?) of SCR.

```
  Bit Supported Command               Command  CCC  Remark
  35  Extension Register Multi-Block  CMD58/59 11   Optional.
  34  Extension Register Single Block CMD48/49 11   Optional.
  33  Set Block Count                  CMD23    2,4  Mandatory for UHS104 card
  32  Speed Class Control              CMD20    2,4  Mandatory for SDXC card
```

If CMD58/59 is supported, then CMD48/49 shall be also supported.

DSi SD/MMC Protocol: PWD Register (128bit Password plus 8bit Password len)

CMD40 - SD - Defined by DPS Spec (Data Protection System) (type=adtc)

```
  Defined by DPS Spec.
```

Response: R1

Additional Data Transfer (to/from whatever):

```
  unknown
```

Single block read type. Intended to read whatever "public" data, which is available even if the card is locked.

CMD42 - SD/MMC (SD v2.00 and up) - SPI - LOCK_UNLOCK (type=adtc)

```
  31-0  Reserved bits (0)
```

Response: R1 (MMC: R1b?)

MMC?: Additional Data Transfer (from card): Busy signal for "R1b" response

Additional Data Transfer (to card):

```
  Note: Before using this command, the size of the following data block (ie.
        "1st..Nth/Extra" byte) must be set via SET_BLOCKLEN command (CMD16).
  1st byte: Flags
    Bit7-4 Reserved (0)
    Bit3   ERASE       Force Erase (1=Erase WHOLE CARD and clear password)
    Bit1   LOCK_UNLOCK Lock card (0=Unlock, 1=Lock) (default on power up: Lock)
    Bit1   CLR_PWD     Clears password  (0=no, 1=yes)
    Bit0   SET_PWD     Set new password (0=no, 1=yes)
  2nd byte: PWDS_LEN   Length of the Password(s) in bytes ("3rd..Nth" byte)
  3rd..Nth byte: Password (old password, if SET_PWD: followed by new password)
  Extra byte: Alignment padding (only in DDR50 mode, if above is odd num bytes)
```

Used to set/clear the password (=to change the password), or to lock/unlock the card (=to log out/log in). If the password exists, then the default state on power-up is Locked (user is logged out).

In locked state, the card will accept only "Basic Commands" (class 0), plus CMD16,40,42, plus ACMD41,42. Observe that this will prevent some initialization commands (for example, switching to 4bit bus via SET_BUS_WIDTH (ACMD6) isn't possible on locked cards).

The password is stored in a 128bit PWD register, so password can be max 16 bytes long. The PWDS_LEN value can be max 32 bytes (when sending old+new password). The length of the current/old password is stored in an 8bit PWD_LEN register (though due to the above limit, that "8bit" value can be in range 0..16 only; PWD_LEN=0 means that there is no password, which is somewhat equivalent to what happens when setting the CLR_PWD bit?).

PWD Notes:

Related CSR status bits are: CARD_IS_LOCKED and LOCK_UNLOCK_FAILED. Presence of the locking feature is indicated by the CCC "class 7" bit in CSD register.

Many SD cards are internally containing separate controller and memory chips, so it might be quite easy to bypass the locking by desoldering one of those chips.

DSi SD/MMC Protocol: State

SD/MMC State

The "state" is an important SD/MMC feature to deal with - most commands can be send only in certain states:
For normal operation, the card should be in "tran" state (and it may then temporarily switch to "data/rcv/prg" states during read/write/erase commands).
For initialization, the card should be first forced to "idle" state, and the init commands should then go through "ready/ident/stby" states, until finally reaching "tran" state.
Less important states are "dis/ina", and, on MMC only, "btst/slp/irq" and "pre-idle".
Both SD and MMC specs are leaving state undocumented for SPI mode (meaning that SPI specific commands like CMD58/CMD59 are supported only in "unknown" state).

SD Card State Transition Table

Command	old state -->	idle	ready	ident	stby	tran	data	rcv	prg	dis	ina
DONE	Operation Complete	----	----	----	----	----	tran	----	tran	stby	----
class 0											
CMD0	GO_IDLE_STATE	ok	idle	idle	idle	idle	idle	idle	idle	idle	----
CMD2	ALL_SEND_CID	----	ident	----	----	----	----	----	----	----	----
CMD3	SEND_RELATIVE_ADDR	----	----	stby	ok	----	----	----	----	----	----
CMD4	SET_DSR	----	----	----	ok	----	----	----	----	----	----
CMD7	SELECT_DESELECT_CARD										
	card is addressed	----	----	----	tran	----	----	----	----	prg	----
	card is not addr.	----	----	----	ok	stby	stby	----	dis	----	----
CMD8	SEND_IF_COND	ok	----	----	----	----	----	----	----	----	----
CMD9	SEND_CSD	----	----	----	ok	----	----	----	----	----	----
CMD10	SEND_CID	----	----	----	ok	----	----	----	----	----	----
CMD11	VOLTAGE_SWITCH	----	ok	----	----	----	----	----	----	----	----
CMD12	STOP_TRANSMISSION	----	----	----	----	----	tran	prg	----	----	----
CMD13	SEND_STATUS	----	----	----	ok	ok	ok	ok	ok	ok	----
CMD15	GO_INACTIVE_STATE	----	----	----	ina	ina	ina	ina	ina	ina	----
class 2											
CMD16	SET_BLOCKLEN	----	----	----	----	ok	----	----	----	----	----
CMD17	READ_SINGLE_BLOCK	----	----	----	----	data	----	----	----	----	----
CMD18	READ_MULTIPLE_BLOCK	----	----	----	----	data	----	----	----	----	----
CMD19	SEND_TUNING_BLOCK	----	----	----	----	data	----	----	----	----	----
CMD20	SPEED_CLASS_CONTROL	----	----	----	----	prg	----	----	----	----	----
CMD23		----	----	----	----	ok	----	----	----	----	----
class 4											
CMD16	SET_BLOCKLEN (2)	----	----	----	----	ok	----	----	----	----	----
CMD20	SPEED_CLASS_CONTROL(2)	----	----	----	----	prg	----	----	----	----	----
CMD23	SET_BLOCK_COUNT	----	----	----	----	ok	----	----	----	----	----
CMD24	WRITE_BLOCK	----	----	----	----	rcv	----	----	----	----	----
CMD25	WRITE_MULTIPLE_BLOCK	----	----	----	----	rcv	----	----	----	----	----
CMD27	PROGRAM_CSD	----	----	----	----	rcv	----	----	----	----	----
class 6											
CMD28	SET_WRITE_PROT	----	----	----	----	prg	----	----	----	----	----
CMD29	CLR_WRITE_PROT	----	----	----	----	prg	----	----	----	----	----
CMD30	SEND_WRITE_PROT	----	----	----	----	data	----	----	----	----	----
class 5											
CMD32	ERASE_WR_BLK_START	----	----	----	----	ok	----	----	----	----	----
CMD33	ERASE_WR_BLK_END	----	----	----	----	ok	----	----	----	----	----
CMD38	ERASE	----	----	----	----	prg	----	----	----	----	----
class 7											
CMD40	Read Block (DPS Spec)	----	----	----	----	data	----	----	----	----	----
CMD42	LOCK_UNLOCK	----	----	----	----	rcv	----	----	----	----	----
class 8											
CMD55	APP_CMD	ok	----	----	ok	ok	ok	ok	ok	ok	----
CMD56	GEN_CMD, RD/WR=0	----	----	----	----	rcv	----	----	----	----	----
	GEN_CMD, RD/WR=1	----	----	----	----	data	----	----	----	----	----
ACMD6	SET_BUS_WIDTH	----	----	----	----	ok	----	----	----	----	----
ACMD13	SD_STATUS	----	----	----	----	data	----	----	----	----	----
ACMD22	SEND_NUM_WR_BLOCKS	----	----	----	----	data	----	----	----	----	----

```
ACMD23 SET_WR_BLK_ERASE_CO. ---- ---- ---- ---- ok   ---- ---- ---- ---- ----
ACMD41 SD_SEND_OP_COND
     OCR check is OK
     and card is not busy   ready---- ---- ---- ---- ---- ---- ---- ---- ----
     OCR check is OK
     and card is busy(2)    ok   ---- ---- ---- ---- ---- ---- ---- ---- ----
     OCR check fails
     query mode             ina  ---- ---- ---- ---- ---- ---- ---- ---- ----
ACMD42 SET_CLR_CARD_DETECT  ---- ---- ---- ---- ok   ---- ---- ---- ---- ----
ACMD51 SEND_SCR             ---- ---- ---- ---- data ---- ---- ---- ---- ----
class 9
class 10 (1)
CMD6  SWITCH_FUNC           ---- ---- ---- ---- data ---- ---- ---- ---- ----
class 11
CMD48 READ_EXTR_SINGLE      ---- ---- ---- ---- data ---- ---- ---- ---- ----
CMD49 WRITE_EXTR_SINGLE     ---- ---- ---- ---- rcv  ---- ---- ---- ---- ----
CMD58 READ_EXTR_MULTI       ---- ---- ---- ---- data ---- ---- ---- ---- ----
CMD59 WRITE_EXTR_MULTI      ---- ---- ---- ---- rcv  ---- ---- ---- ---- ----
ACMD14-16               Refer to DPS Specification (class 8)
ACMD28                  Refer to DPS Specification (class 8)
ACMD18,25,26,38,
43,44,45,46,47,48,49    Refer to the "Part3 Security Specification" for
                        information about the SD Security Features (class 8)
CMD52-CMD54             Refer to the "SDIO Card Specification" (class 9)
CMD21                   Refer to DPS Specification (class 11)
CMD34-37,50,57          Refer to each command system specification (class 10)
CMD41,CMD43-47          reserved (class 11)
CMD60...CMD63           reserved for manufacturer (class 11)
SPI Mode
CMD1 SEND_OP_COND       SPI-only
CMD58 READ_OCR          SPI-only
CMD59 CRC_ON_OFF        SPI-only
```

Note (1): Class 10 commands were defined in Version 1.10.

Note (2): Card returns busy in case of following:

```
- Card executes internal initialization process
- When HCS in the argument is set to 0 to SDHC or SDXC Card.
```

The state transitions of the SD Memory Card application-specific commands are given under Class 8, above.

```
----  command is treated as illegal command
ok    command is accepted, and card stays in SAME state
xxx   command is accepted, and card switches to "xxx" state
```

MMC Card State Transition Table (JEDEC)

```
Command     old state --> idl rdy idt stb trn dta tst rcv prg dis ina slp irq
Class Independent
ERR  CRC error            --- --- --- --- --- --- --- --- --- --- --- --- stb
ERR  command not supported--- --- --- --- --- --- --- --- --- --- --- --- stb
Class 0
CMD0  (arg=00000000h)     ok  idl idl idl idl idl idl idl idl idl --- idl stb
      GO_IDLE_STATE
CMD0  (arg=F0F0F0F0h)     pre pre pre pre pre pre pre pre pre pre --- pre stb
      GO_PRE_IDLE_STATE
CMD0  (arg=FFFFFFFAh)     initiate alternative boot operation
      BOOT_INITIATION
CMD1  SEND_OP_COND
      card VDD range ok   rdy --- --- --- --- --- --- --- --- --- --- --- stb
      card is busy        ok  --- --- --- --- --- --- --- --- --- --- --- stb
      card VDD range bad  ina --- --- --- --- --- --- --- --- --- --- --- stb
CMD2  ALL_SEND_CID
       card wins bus      --- idt --- --- --- --- --- --- --- --- --- --- stb
       card loses bus     --- ok  --- --- --- --- --- --- --- --- --- --- stb
CMD3  SET_RELATIVE_ADDR   --- --- stb --- --- --- --- --- --- --- --- --- stb
CMD4  SET_DSR             --- --- --- ok  --- --- --- --- --- --- --- --- stb
CMD5  SLEEP_AWAKE         --- --- --- slp -?- -?- -?- -?- -?- -?- -?- stb stb
CMD6  SWITCH              --- --- --- --- prg --- --- --- --- --- --- --- stb
```

CMD7	SELECT_DESELECT_CARD													
	card is addressed	---	---	---	trn	---	---	---	---	---	prg	---	---	stb
	card is not addr.	---	---	---	---	stb	stb	---	---	dis	---	---	---	stb
CMD8	SEND_EXT_CSD	---	---	---	---	dta	---	---	---	---	---	---	---	stb
CMD9	SEND_CSD	---	---	---	ok	---	---	---	---	---	---	---	---	stb
CMD10	SEND_CID	---	---	---	ok	---	---	---	---	---	---	---	---	stb
CMD12	STOP_TRANSMISSION	---	---	---	---	---	trn	---	prg	---	---	---	---	stb
CMD13	SEND_STATUS	---	---	---	ok	ok	ok	ok	ok	ok	ok	---	---	stb
CMD14	BUSTEST_R	---	---	---	---	---	---	trn	---	---	---	---	---	stb
CMD15	GO_INACTIVE_STATE	---	---	---	ina	ina	ina	ina	ina	ina	ina	---	---	stb
CMD19	BUSTEST_W	---	---	---	---	tst	---	---	---	---	---	---	---	stb
Class 1														
CMD11	READ_DAT_UNTIL_STOP	---	---	---	---	dta	---	---	---	---	---	---	---	stb
Class 2														
CMD16	SET_BLOCKLEN	---	---	---	---	ok	---	---	---	---	---	---	---	stb
CMD17	READ_SINGLE_BLOCK	---	---	---	---	dta	---	---	---	---	---	---	---	stb
CMD18	READ_MULTIPLE_BLOCK	---	---	---	---	dta	---	---	---	---	---	---	---	stb
CMD23	SET_BLOCK_COUNT	---	---	---	---	ok	---	---	---	---	---	---	---	stb
Class 3														
CMD20	WRITE_DAT_UNTIL_STOP	---	---	---	---	rcv	---	---	---	---	---	---	---	stb
Class 4														
CMD16	SET_BLOCKLEN	see class 2												
CMD23	SET_BLOCK_COUNT	see class 2												
CMD24	WRITE_BLOCK	---	---	---	---	rcv	---	---	---	rcv1	---	---	---	stb
CMD25	WRITE_MULTIPLE_BL.	---	---	---	---	rcv	---	---	---	rcv2	---	---	---	stb
CMD26	PROGRAM_CID	---	---	---	---	rcv	---	---	---	---	---	---	---	stb
CMD27	PROGRAM_CSD	---	---	---	---	rcv	---	---	---	---	---	---	---	stb
Class 6														
CMD28	SET_WRITE_PROT	---	---	---	---	prg	---	---	---	---	---	---	---	stb
CMD29	CLR_WRITE_PROT	---	---	---	---	prg	---	---	---	---	---	---	---	stb
CMD30	SEND_WRITE_PROT	---	---	---	---	dta	---	---	---	---	---	---	---	stb
CMD31	SEND_WRITE_PROT_TYPE	---	---	---	---	dta	---	---	---	---	---	---	---	stb
Class 5														
CMD35	ERASE_GROUP_START	---	---	---	---	ok	---	---	---	---	---	---	---	stb
CMD36	ERASE_GROUP_END	---	---	---	---	ok	---	---	---	---	---	---	---	stb
CMD38	ERASE	---	---	---	---	prg	---	---	---	---	---	---	---	stb
Class 7														
CMD16	SET_BLOCKLEN	see class 2												
CMD42	LOCK_UNLOCK	---	---	---	---	rcv	---	---	---	---	---	---	---	stb
Class 8														
CMD55	APP_CMD	---	---	---	ok	ok	ok	ok	ok	ok	ok	---	---	ok
CMD56	GEN_CMD, RD/WR=0	---	---	---	---	rcv	---	---	---	---	---	---	---	stb
	GEN_CMD, RD/WR=1	---	---	---	---	dta	---	---	---	---	---	---	---	stb
Class 9														
CMD39	FAST_IO	---	---	---	ok	---	---	---	---	---	---	---	---	stb
CMD40	GO_IRQ_STATE	---	---	---	irq	---	---	---	---	---	---	---	---	stb
Class 10-11														
CMD41, CMD43..CMD54		Reserved												
CMD57..CMD59		Reserved												
CMD60..CMD63		Reserved for Manufacturer												
SPI Mode														
CMD58	READ_OCR	SPI-only												
CMD59	CRC_ON_OFF	SPI-only												

NOTE 1. Due to legacy considerations, a card may treat CMD24/25 during a prg state_while busy is active_as a legal or an illegal command. A card that treats CMD24/25 during a prg-state_while busy is active_as an illegal command will not change its state to the rcv state. A host should not send CMD24/25 while the card is in prg state and busy is active.

NOTE 2. Due to legacy considerations, a card may treat CMD24/25 during a prg state_while busy is active_as a legal or an illegal command. A card that treats CMD24/25 during a prg state_while busy is active_as an illegal command will not change its state to the rcv state. A host should not send CMD24/25 while the card is in prg state and busy is active.

NOTE 3. As there is no way to obtain state information in boot mode, boot-mode states are not shown in this table.

```
  pre  Pre-idle
  idl  idle
  rdy  ready
  idt  ident
  stb  stby
  trn  tran
  dta  data
  tst  btst
```

DSi SD/MMC Protocol: Signals

XXX...

SD Mode 1-bit data transfer mode

```
          __ start bit                              __ checksum bits (CRC-CCITT)
         |                                         |
         |  <------------data bits------------->   |   __ stop bit
         |                                         |  |
  DAT0   0  1st 2nd 3rd 4th 5th 6th 7th ... last crc 1
```

SD Mode 4-bit data transfer mode

```
          __ start bit          __ checksum bits (CRC-CCITT)
         |                     |
         |  <--data bits-->    |   __ stop bit
         |                     |  |
  DAT3   0  1st 5th ... ...   crc 1
  DAT2   0  2nd 6th ... ...   crc 1
  DAT1   0  3rd 7th ... ...   crc 1
  DAT0   0  4th 8th ... last crc 1
```

DSi SDIO Special SDIO Commands

CMD52 - SDIO: IO_RW_DIRECT

Read/write single byte. Mostly used for detection/configuration via SDIO Function 0 commands.

```
  31     R/W Flag               (0=Read, 1=Write)
  30-28  Function Number (3bit)
  27     Read-after-write (RAW) Flag (if Bit31=1=Write, and Bit27=1)
  26     Stuff (unspecified, should be probably 0, but is 1 on DSi)
  25-9   Register Address (17bit)
  8      Stuff (unspecified, should be probably 0, but is 1 on DSi)
  7-0    Write Data (8bit), or Stuff bits (for read)
```

SD Mode Response: R5:

```
  47      Start Bit (0)                                  ;\
  46      Transmission To Host (0)                       ; 1st byte
  45-40   Command (the 6bit CMD being responded to)      ;/
  39-24   Stuff Bits                                     ;-2nd..3rd byte
  23-16   Response Flags                                 ;-4th byte
           7   COM_CRC_ERROR
           6   ILLEGAL_COMMAND
           5-4 IO_CURRENT STATE (0=dis, 1=cmd, 2=trn(cmd53), 3=rfu)
           3   ERROR
           2   RFU (reserved for future use)
           1   INVALID_FUNCTION_NUMBER
           0   OUT_OF_RANGE
  15-8    Read or Write Data (8bit)                      ;-5th byte
  7-1     CRC7                                           ;\6th byte
  0       End Bit (1)                                    ;/
```

SPI Mode Response: R5:

```
  8bit  modified R1 response
          7  start bit (0)
          6  parameter error       (0=okay, 1=error)
          5  RFU (0)
          4  function number error (0=okay, 1=error)
          3  COM CRC error         (0=okay, 1=error)
          2  illegal command       (0=okay, 1=error)
          1  RFU (0)
          0  in idle state         (0=no, 1=idle)
  8bit  Read or Write Data
```

CMD53 - SDIO: IO_RW_EXTENDED

Mostly used for actual command/data transfers via SDIO Function 1 commands.

```
  31     R/W Flag               (0=Read, 1=Write)
  30-28  Function Number (3bit) (0=CIA)
  27     Block Mode             (0=Bytes, 1=Blocks/optional)
  26     OP Code                (0=Fixed Address, 1=Incrementing Address)
  25-9   Register Address (17bit)
  8-0    Byte/Block Count (9bit) (1..511) (0=512 Bytes, or 0=Infinite Blocks)
```

Response: R5: Same as for CMD52 (with 8bit data = 00h)

Data Transfer:

```
  For Byte Mode: Similar to CMD17/CMD24 (single block)
  For Block Mode: Similar to CMD18/CMD25 (multiple block)
  For Block Mode: CMD52:STOP_TRANSMISSION only needed if using "InfiniteBlocks"
```

CMD5 - SPI - SDIO: IO_SEND_OP_COND

Similar to SD Memory Card's ACMD41.

```
  31-25  stuff bits (0)
  24     Switching to 1.8V Request (S18R)
  23     I/O OCR VDD Voltage Window 3.5V-3.6V
  22     I/O OCR VDD Voltage Window 3.4V-3.5V
  21     I/O OCR VDD Voltage Window 3.3V-3.4V
  20     I/O OCR VDD Voltage Window 3.2V-3.3V
  19     I/O OCR VDD Voltage Window 3.1V-3.2V
  18     I/O OCR VDD Voltage Window 3.0V-3.1V
  17     I/O OCR VDD Voltage Window 2.9V-3.0V
  16     I/O OCR VDD Voltage Window 2.8V-2.9V
  15     I/O OCR VDD Voltage Window 2.7V-2.8V
  14     I/O OCR VDD Voltage Window 2.6V-2.7V
  13     I/O OCR VDD Voltage Window 2.5V-2.6V
  12     I/O OCR VDD Voltage Window 2.4V-2.5V
  11     I/O OCR VDD Voltage Window 2.3V-2.4V
  10     I/O OCR VDD Voltage Window 2.2V-2.3V
  9      I/O OCR VDD Voltage Window 2.1V-2.2V
  8      I/O OCR VDD Voltage Window 2.0V-2.1V
  7-4    I/O OCR VDD Voltage Window Reserved
  3-0    I/O OCR VDD Voltage Window Reserved
```

SD Mode Response: R4:

```
  47     Start Bit (0)                                ;\
  46     Transmission To Host (0)                     ; 1st byte
  45-40  Reserved (111111) (instead of Command value) ;/
  39     Card is ready to operate after init          ;\
  38-36  Number of I/O Functions                      ;
  35     Memory Present                               ; 2nd byte
  34-33  Stuff bits (0)                               ;
  32     Switching to 1.8V Accepted (S18R) (not SPI)  ;/
  31-8   I/O OCR (24bit)                              ;-3rd..5th byte
  7-1    Reserved (111111) (instead of CRC7)          ;\6th byte
  0      End Bit (1)                                  ;/
```

SPI Mode Response: R4:

```
  8bit   modified R1 Response
          7  start bit (0)
          6  parameter error       (0=okay, 1=error)
```

```
          5  RFU (0)
          4  function number error (0=okay, 1=error)
          3  COM CRC error         (0=okay, 1=error)
          2  illegal command       (0=okay, 1=error)
          1  RFU (0)
          0  in idle state         (0=no, 1=idle)
  32bit  same as SD Response bit39-8 (but without S18R bit)
```

DSi SDIO Wifi Init

Related required registers/bits are:

```
 - SCFG_EXT7.bit19 needed for SDIO controller (else 4004Axxh-4004Bxxh disabled)
 - SCFG_CLK7 seems to be NOT needed for SDIO clock enable (unlike SDMMC)
 - SCFG_WL.bit0 seems to be wifi-related (but effect is unknown)
 - GPIO_WIFI.bit8 needed for AR6013G chips (else SDIO Function 1 fails)
 - BPTWL[30h] needed for LED and SDIO (else SDIO fails badly)
 - RTC.FOUT pin as configured by firmware (else WMI commands/events fail)
```

DSi init sequence is trying to send one CMD52 command first; if that fails, then the DSi is sending several CMD5's, followed by CMD3+CMD7.

SDIO State

```
  Command                 ini  stb  cmd  trn  ina
  CMD3  SET_RELATIVE_ADDR stb  ok   ---  ---  ---
  CMD5  IO_SEND_OP_COND   ok   ---  ---  ---  ---
        ocr bad           ina  ---  ---  ---  ---
  CMD7  SELECT_CARD       ---  cmd  ok   ---  ---
        DESELECT_CARD     ---  ok   stb  ---  ---
  CMD15 GO_INACTIVE_STATE ina  ina  ina  ---  ---
  CMD52 IO_RW_DIRECT      ---  ---  ok   (cmd)---
  CMD53 IO_RW_EXTENDED    ---  ---  trn  ---  ---
```

Note: In CMD52, state "dis" can mean state "ini","stb","ina" (though, theoretically CMD52 cannot be used in that states, so one should never see the "dis" state at all).

More SD commands that are (occassionally) used for SDIO

```
  CMD0  GO_IDLE_STATE     for entering SPI mode only (but does NOT reset SDIO)
  CMD8  SEND_IF_COND      optional for SDHC/SDXC
  CMD11 VOLTAGE_SWITCH    optional for UHS-I
  CMD19 SEND_TUNING_BLOCK optional for UHS-I
  CMD59 CRC_ON_OFF        spi-only
```

Moreover a combo card (a SDIO device with built-in SD memory card) may implement various SD commands; these commands will affect only the SD memory card side, not the SDIO device).

SDIO doesn't have CID or CSD registers, nor commands for STOP_TRANSMISSION, SET_BUSWIDTH, or SET_BLOCKLEN (but CMD52 can do equivalent stuff via SDIO Function 0 registers).

_____ I/O Commands for MMC_____

CMD39 - MMCIO: FAST_IO (type=ac)

```
  31-16  RCA
  15     Register Write Flag
  14-8   Register Address
  7-0    Register Data
```

MMC Response: R4:

```
  47      Start Bit (0)                                ;\
  46      Transmission To Host (0)                     ; 1st byte
  45-40   Command (the 6bit CMD being responded to)    ;/
  39-24   RCA                                          ;-2nd..3rd byte
  23      Status (0=Bad, 1=Successful)                 ;\4th byte
  22-16   Register Address                             ;/
  15-8    Read Register Contents                       ;-5th byte
  7-1     CRC7                                         ;\6th byte
  0       End Bit (1)                                  ;/
```

CMD40 - MMCIO: GO_IRQ_STATE (type=bcr)

```
  31-0   Stuff Bits
```

MMC Response: R5:

```
  47      Start Bit (0)                                  ;\
  46      Transmission To Host (0)                       ; 1st byte
  45-40   Command (the 6bit CMD being responded to)      ;/
  39-24   RCA                                            ;-2nd..3rd byte
  23-8    Not defined (may be used for IRQ data)         ;-4th..5th byte
  7-1     CRC7                                           ;\6th byte
  0       End Bit (1)                                    ;/
```

DSi SDIO Memory and I/O Maps

Function 0 - Common I/O Area (CIA)

```
  0:00000h..000FFh  Card Common Control Registers (CCCR)
  0:00100h..001FFh  Function Basic Registers (FBR) for Function 1
  0:00200h..002FFh  Function Basic Registers (FBR) for Function 2
  0:00300h..003FFh  Function Basic Registers (FBR) for Function 3
  0:00400h..004FFh  Function Basic Registers (FBR) for Function 4
  0:00500h..005FFh  Function Basic Registers (FBR) for Function 5
  0:00600h..006FFh  Function Basic Registers (FBR) for Function 6
  0:00700h..007FFh  Function Basic Registers (FBR) for Function 7
  0:00800h..00FFFh  Reserved for Future
  0:01000h..17FFFh  Card Information Structures (Common CIS and Func 1-7 CIS)
  0:18000h..1FFFFh  Reserved for Future
```

Function 1..7 - Function specific Register Space

```
  n:00000h..1FFFFh  Registers (seven 128K spaces, one for each function)
```

Code Storage Area (CSA) (optional, R or R/W)

```
  CSA:00000h..FFFFFh  16Mbyte FAT12/FAT16 (accessed indirectly via "Window")
```

Card Common Control Registers (CCCR)

```
  0:00000h 2   CCCR: Revision (R)
  0:00002h 2   CCCR: I/O Function Enable/Ready (R/W)
  0:00004h 2   CCCR: Interrupt Enable/Pending (R/W)
  0:00006h 1   CCCR: I/O Abort (W)
  0:00007h 1   CCCR: Bus Interface Control (R/W)
  0:00008h 1   CCCR: Card Capability
  0:00009h 3   CCCR: Common CIS Pointer, Lo/Mid/Hi
  0:0000Ch 1   CCCR: Bus Suspend
  0:0000Dh 1   CCCR: Function Select (R/W)
  0:0000Eh 2   CCCR: Exec/Ready Flags (R)
  0:00010h 2   CCCR: CMD53 Block Size for Function 0, Lo/Hi (R/W)
  0:00012h 1   CCCR: Power Control
  0:00013h 2   CCCR: Bus Speed Select
  0:00015h 1   CCCR: Driver Strength
  0:00016h 1   CCCR: Interrupt Extension
  0:00017h D9h CCCR: Reserved for Future
  0:000F0h 10h CCCR: Reserved for Vendors
```

Function Basic Registers (FBR) for Function n (n=1..7)

```
  0:00n00h 1    FBR(n): Misc
  0:00n01h 1    FBR(n): Extended standard SDIO Function interface code
  0:00n02h 1    FBR(n): Misc
  0:00n02h 7    FBR(n): Reserved for Future
  0:00n09h 3    FBR(n): Pointer to Card Information Structure (CIS), Lo/Mid/Hi
  0:00n0Ch 3    FBR(n): Code Storage Area (CSA) Address, Lo/Mid/Hi
  0:00n0Fh 1    FBR(n): Code Storage Area (CSA) Data "Window"
```

```
  0:00n10h 2    FBR(n): CMD53 Block Size for Function n, Lo/Hi
  0:00n12h EEh  FBR(n): Reserved for Future
```

DSi SDIO Common Control Registers (CCCR)

0:00000h - CCCR: Revision (R)

```
  0-3   CCCR/FBR Format Version      (0=v1.00, 1=v1.10, 2=v2.00, 3=v3.00) (R)
  4-7   SDIO Spec Version   (0=v1.00, 1=v1.10, 2=v1.20, 3=v2.00, 4=v3.00) (R)
  8-11  SD Physical Layer Spec       (0=v1.01, 1=v1.10, 2=v2.00, 3=v3.0x) (R)
  12-15 Reserved for Future                                               (-)
```

0:00002h - CCCR: I/O Function Enable/Ready (R/W)

```
  0     Reserved for Future                                               (-)
  1-7   SDIO Function 1..7 Enable Flags            (0=Disable, 1=Enable) (R/W)
  8     Reserved for Future                                               (-)
  9-15  SDIO Function 1..7 Ready Flags         (0=Disabled/Busy, 1=Ready) (R)
```

0:00004h - CCCR: Interrupt Enable/Pending (R/W)

```
  0     SDIO Interrupt Master Enable               (0=Disable, 1=Enable) (R/W)
  1-7   SDIO Function 1..7 Interrupt Enable        (0=Disable, 1=Enable) (R/W)
  8     Reserved for Future                                               (-)
  9-15  SDIO Function 1..7 Interrupt Pending               (0=No, 1=IRQ) (R)
```

0:00006h - CCCR: I/O Abort (W)

```
  0-2   SDIO Function Number to be Aborted  (0=None?, 1..7=Function 1..7) (W)
          XXXsee pg 35
  3     Reset SDIO Card                              (0=Normal, 1=Reset) (W)
  4-7   Reserved for Future                                               (-)
```

0:00007h - CCCR: Bus Interface Control (R/W)

```
  0-1   Bus Width       (0=1bit, 1=Reserved, 2=4bit, 3=EmbeddedSDIO/8bit) (R/W)
  2     Support 8bit Bus Flag            (0=No, 1=Yes/EmbeddedSDIO only) (R)
  3-4   Reserved for Future                                               (-)
  5     Enable Continous SPI Interrupt             (0=Disable, 1=Enable) (R/W)
  6     Support Continous SPI Interrupt                    (0=No, 1=Yes) (R)
  7     Card Detect Disable     (0=Enable Pull-up on DAT3 pin, 1=Disable) (R/W)
```

0:00008h - CCCR: Card Capability

```
  0     Support Direct Command (CMD52) during Data Transfer (0=No, 1=Yes) (R)
  1     Support Multi-Block transfer (CMD53.block mode)     (0=No, 1=Yes) (R)
  2     Support Read Wait Control (RWC via DAT2 pin)        (0=No, 1=Yes) (R)
  3     Support Bus Control Suspend/Resume                  (0=No, 1=Yes) (R)
  4     Support Block Gap Interrupt during Multi-Block      (0=No, 1=Yes) (R)
  5     Enable Block Gap Interrupt during Multi-Block    (0=No, 1=Enable) (R/W)
  6     Low Speed Card                        (0=Full-Speed, 1=Low-Speed) (R)
  7     Support 4bit Mode for Low-Speed Card                (0=No, 1=Yes) (R)
```

0:00009h - CCCR: Common CIS Pointer, Lo
0:0000Ah - CCCR: Common CIS Pointer, Mid
0:0000Bh - CCCR: Common CIS Pointer, Hi

```
  0-16  Pointer to Card Common Card Information Structure (Common CIS)    (R)
  17-23 Unspecified (probably reserved)                                   (-)
```

0:0000Ch - CCCR: Bus Suspend

```
  0     Bus Status             XXX see pg 37                              (R)
  1     Bus Release Request    XXX see pg 38                              (R)
  2-7   Reserved for Future                                               (-)
```

0:0000Dh - CCCR: Function Select (R/W)

```
  0-3   Select Function (0=CIA, 1..7=Function 1..7, 8=Memory Card)        (R/W)
  4-6   Reserved for Future                                               (-)
  7     Data Flag (more data after resuming) (0=No, 1=Yes)                (R)
```

0:0000Eh - CCCR: Exec/Ready Flags (R)

```
  0     Command Execution Flag for Memory (=SD/Combo? or CSA?)            (R)
  1-7   Command Execution Flags for Function 1..7       (0=Busy, 1=Ready) (R)
  8     Read/Write Ready Flag for Memory (=SD/Combo? or CSA?)             (R)
  9-15  Read/Write Ready Flags for Function 1..7        (0=Busy, 1=Ready) (R)
```

0:00010h - CCCR: CMD53 Block Size for Function 0, Lo (R/W)
0:00011h - CCCR: CMD53 Block Size for Function 0, Hi (R/W)

```
  0-15  CMD53 Block size for Function(0)          (0001h..0800h) (0=None) (R/W)
```

0:00012h - CCCR: Power Control

```
  0     Support Master Power Control                          (0=No, 1=Yes) (R)
  1     Enable Master Power Control    (0=No/max 720mW, 1=Yes/allow more) (R/W)
  2-7   Reserved for Future                                               (-)
```

0:00013h,00014h - CCCR: Bus Speed Select

```
  0     Support High-Speed Mode (SDR25 or higher)             (0=No, 1=Yes) (R)
  1-3   Bus Speed Select   (0=SDR12, 1=SDR25, 2=SDR50, 3=SDR104, 4=DDR50) (R/W)
  4-7   Reserved for Future                                               (-)
  8     Support UHS-I SDR50  (usable in 1.8V mode only)       (0=No, 1=Yes) (R)
  9     Support UHS-I SDR104 (usable in 1.8V mode only)       (0=No, 1=Yes) (R)
  10    Support UHS-I DDR50  (usable in 1.8V mode only)       (0=No, 1=Yes) (R)
  11-15 Reserved for Future
```

0:00015h - CCCR: Driver Strength

```
  0     Support Driver Type A ;\see Physical Layer Specs      (0=No, 1=Yes) (R)
  1     Support Driver Type C ; version 3.0x for details      (0=No, 1=Yes) (R)
  2     Support Driver Type D ;/                              (0=No, 1=Yes) (R)
  3     Reserved for Future                                               (-)
  5-4   Driver Type Select    (0=Default/B, 1=Type A, 2=Type C, 3=Type D) (R/W)
  7-6   Reserved for Future                                               (-)
```

0:00016h - CCCR: Interrupt Extension

```
  0     Support Asynchronous Interrupt in 4bit mode           (0=No, 1=Yes) (R)
  1     Enable Asynchronous Interrupt in 4bit mode         (0=No, 1=Enable) (R/W)
  7-2   Reserved for Future                                               (-)
```

DSi SDIO Function Basic Registers (FBR)

0:00n00h - FBR(n): Interface Type

```
  0-3   Standard SDIO Function Interface Code                             (R)
  4-5   Reserved for Future                                               (-)
  6     Code Storage Area (CSA) Supported                     (0=No, 1=Yes) (R)
  7     Code Storage Area (CSA) Enable   (0=Block reads/writes, 1=Enable) (R/W)
  8-15  Extended standard SDIO Function interface code  (when bit0-3=0Fh) (R)
```

The interface type is merged of bit0-3 and bit8-15:

```
  0h:00h = No SDIO standard interface (eg. Atheros Wifi in DSi)
  1h:00h = SDIO Standard UART
  2h:00h = SDIO Bluetooth Type-A standard interface
  3h:00h = SDIO Bluetooth Type-B standard interface
  4h:00h = SDIO GPS standard interface
  5h:00h = SDIO Camera standard interface
  6h:00h = SDIO PHS standard interface
  7h:00h = SDIO WLAN interface
  8h:00h = Embedded SDIO-ATA standard interface
```

```
  9h:00h = SDIO Bluetooth Type-A Alternate MAC PHY (AMP) standard interface
  Ah:00h = Reserved for Future
  Bh:00h = Reserved for Future
  Ch:00h = Reserved for Future
  Dh:00h = Reserved for Future
  Eh:00h = Reserved for Future
  Fh:00h..FFh = Reserved for Future
```

0:00n02h - FBR(n): Power

```
  0     Support Power Selection                              (0=No, 1=Yes) (R)
  1     Enable Power Selection        (0=Normal Current, 1=Lower Current) (R/W)
  2-3   Reserved for Future                                                (-)
  4-7   Power State                                                        (R/W)
```

0:00n09h-00n0Bh - FBR(n): Pointer to Card Information Structure (CIS)

```
  0-16  Pointer to Function(n)'s Card Information Structure (Function CIS)(R)
  17-23 Unspecified (probably reserved)                                    (-)
```

Should point to "End-of-Chain Tuple" for unsupported functions?

0:00n0Ch-00n0Eh - FBR(n): Code Storage Area (CSA) Address (R/W)

```
  0-23  Pointer to CSA memory (incremented after CSA data read/write)     (R/W)
```

0:00n0Fh - FBR(n): Code Storage Area (CSA) Data "Window" (R or R/W)

```
  0-7   Data (to/from auto-incrementing CSA Address) (R for ROM, R/W otherwise)
```

0:00n10h-00n11h - FBR(n): CMD53 Block Size (R/W)

```
  0-15  CMD53 Block size for Function(n)          (0001h..0800h) (0=None) (R/W)
```

DSi SDIO Card Information Structures (CIS)

The CIS used by SDIO is based directly upon the metaformat specification used by PCMCIA and Compact Flash. For details on the metaformat, refer to:

```
  PC Card Standard, Volume 4, Metaformat Specification
```

Published by: PCMCIA (Personal Computer Memory Card International Association)

Basic Format

```
  00h           CISTPL_code
  01h           Offset to next tuple (n) (aka size of body)
  02h+(0..n-1)  Body (n bytes)
```

Summary

```
  00h = CISTPL_NULL          Null Tuple
  10h = CISTPL_CHECKSUM      Checksum Control
  15h = CISTPL_VERS_1        Level 1 Version/Product Information
  16h = CISTPL_ALTSTR        Alternate Language String
  20h = CISTPL_MANFID        Manufacturer ID
  21h = CISTPL_FUNCID        Function ID
  22h = CISTPL_FUNCE         Function Extensions
  80h-8Fh = Vendor specific  Vendor specific
  91h = CISTPL_SDIO_STD      Info for Standard SDIO Functions
  92h = CISTPL_SDIO_EXT      Reserved for future SDIO stuff
  FFh = CISTPL_END           End-of-chain
```

Tuple 00h - Null Tuple (meaningless, unknown purpose)

```
  00h Tuple ID   (00h)
  01h Tuple Size (00h)
```

Unspecified in SDIO reference... but maybe defined in PCMIA specs?)

Tuple 10h - Checksum Control

```
  00h Tuple ID   (10h)
  01h Tuple Size (?)
  ... Unknown
```

Unspecified in SDIO reference... but maybe defined in PCMIA specs?)

Tuple 15h - Level 1 Version/Product Information

```
  00h Tuple ID   (15h)
  01h Tuple Size (?)
  ... Unknown
```

Unspecified in SDIO reference... but maybe defined in PCMIA specs?)

Tuple 20h - Manufacturer ID

```
  00h      Tuple ID   (20h)
  01h      Tuple Size (at least 4)
  02h-03h  Manufacturer ID (assigned by JEIDA or PCMCIA)
  04h-05h  Part Number/Revision (manufacturer specific)
```

Tuple 21h - Function ID

```
  00h      Tuple ID   (21h)
  01h      Tuple Size (2)
  02h      Card Function Code (0Ch for SDIO)
  03h      System initialization bit mask (Not used, 00h)
```

Tuple 22h - Function Extension

```
  00h      Tuple ID   (22h)
  01h      Tuple Size (..)
  02h      Type of extended data
  03h..xxh Function information
```

Tuple 22h - Function Extension, Type 00h, for Function 0

```
  00h      Tuple ID   (22h)
  01h      Tuple Size (04h+2*N)
  02h      Type of extended data (00h=Type 00h)
  03h-04h  Max Block Size for Function 0 (0001h or higher)
  05h      Max Transfer Speed for Function 0-7 (specified as Value*Unit bits/s)
           bit0-2: Unit  (0=0.1M, 1=1M, 2=10M, 3=100M, 4..7=Reserved)
           bit3-6: Value (0=Reserved, 1=1, 2=1.2, 3=1.3, 4=1.5, 5=2, 6=2.5,
                   7=3, 8=3.5, 9=4, 10=4.5, 11=5, 12=5.5, 13=6, 14=7, 15=8)
           bit7:   Reserved
  06h...   N two-byte pairs (TC,CP) for 1..N   ;(N=([01h]-4)/2)
```

Tuple 22h - Function Extension, Type 01h, for Function 1-7

```
  00h      Tuple ID   (22h)
  01h      Tuple Size (2Ah)
  02h      Type of extended data (01h=Type 01h)
  03h      Function Info (bit0=WakeUpSupport, bit1..7=Reserved)
  04h      Standard SDIO Function version (2x4bit maj.min, or 00h=Nonstandard)
  05h-08h  Card Product Serial Number PSN (32bit) (unique value, or 0=None)
  09h-0Ch  CSA Size in bytes available for this Function (32bit)
  0Dh      CSA Property (bit0=WriteProtected/ReadOnly, bit1=NoReformatting)
  0Eh-0Fh  Max Block Size for this Function (0001h or higher)
  10h-13h  Operation Condition OCR (same as in ACMD41 for SD Memory devices)
  14h-16h  3x8bit Operation Power (Min/Average/Max) (0..254mA, or 255=more)
  17h-19h  3x8bit Standby Power   (Min/Average/Max) (0..254mA, or 255=more)
  1Ah-1Dh  2x16bit Bandwidth      (Min/Optimal) (1..65535 KB/sec, or 0=None)
  1Eh-1Fh  Timeout for Enable-till-Ready in 10ms units (max 655.35 seconds)
  20h-23h  2x16bit Operation 3.3V         (Average/Max) (1..65535mA, or 0=?)
  24h-25h  2x16bit High-Current-Mode 3.3V (Average/Max) (1..65535mA, or 0=?)
  28h-2Bh  2x16bit Low-Current-Mode 3.3V  (Average/Max) (1..65535mA, or 0=?)
```

Tuple 22h - Function Extension, Type 02h, for Function 1-7 (Power State)

```
  00h      Tuple ID   (22h)
  01h      Tuple Size (02h+N*2) (N=1..15, for up to 15 power states)
  02h      Type of extended data (02h=Type 02h)
  03h      Fixed value (00h)
  04h..xxh Nx16bit Max consumption in Power State 1..N (0..65535mW)
```

Tuple 91h - SDIO_STD - Info about Standard SDIO Functions

```
  00h      Tuple ID   (91h)
  01h      Tuple Size (02h..FFh)
  02h      SDIO STD ID (the 4+8bit Interface Type in FBR, squeezed into 8bits?)
  03h      SDIO STD Type           ;\depends on Interface Type
  04h...   SDIO STD Data (if any)  ;/
```

Tuple 92h - SDIO_EXT

```
  00h      Tuple ID   (92h)
  01h      Tuple Size (?)
  02h...   Reserved (if any)
```

Tuple FFh - END

```
  00h      Tuple ID   (FFh)
```

Indicates the end of tuple's list. Unlike all other tuple's, this is only a single FFh-byte (without "Tuple Size" entry).

01 CISTPL_DEVICE
14 CISTPL_NO_LINK
1A CISTPL_CONFIG
1B CISTPL_CFTABLE_ENTRY

DSi SD/MMC Filesystem

DSi Partition Table and FAT Filesystems

DSi SD/MMC Partition Table (aka Master Boot Record aka MBR)
DSi SD/MMC Filesystem (FAT)

System Tools and DSiware games are having regular "ROM Cartridge" headers, with NitroROM filesystem defined in that headers (ie. the ".app" files are internally containing a 2nd filesystem inside of the FAT filesystem; eventually there can be also NARC files as 3rd filesystem inside of the NitroROM filesystem).

DSi Cartridge Header
DS Cartridge NitroROM and NitroARC File Systems

Savedata for DSiWare games is usually stored in "public.sav" or "private.sav" files. That .sav files are usually containing a FAT12 with its own VBR, FAT, and Directories (so they use some virtual FAT12 inside of the real FAT16).

DSi Filesystem Overview

DSi SD/MMC Internal NAND Layout
DSi SD/MMC Bootloader
DSi SD/MMC Device List
DSi SD/MMC Complete List of SD/MMC Files/Folders
DSi SD/MMC Summary of SD/MMC Files/Folders
DSi SD/MMC Images
DSi Autoload on Warmboot

DSi Files

DSi SD/MMC DSiware Files on Internal eMMC Storage

DSi SD/MMC Partition Table (aka Master Boot Record aka MBR)

DSi eMMC Partition table

The decrypted DSi MBR contains (for 240MB chips; 3rd part differs for 245.5MB):

```
  0000  00 00 00 00 00 00 00 00 .. .. .. .. 00 00        ;bootcode (zero)
  01BE  00 03 18 04 06 0F E0 3B 77 08 00 00 89 6F 06 00  ;1st partition (main)
  01CE  00 02 CE 3C 06 0F E0 BE 4D 78 06 00 B3 05 01 00  ;2nd partition (photo)
  01DE  00 02 DE BF 01 0F E0 BF 5D 7E 07 00 A3 01 00 00  ;3rd partition (extra)
  01EE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ;4th partition (none)
  01FE  55 AA                                            ;mbr id (55h,AAh)
```

Above values are following the classical standard MBR format:

```
  000h      446  bootcode (zerofilled on DSi)                  ;-bootcode
  1BEh+n*10h  1  status (00h)                                   ;\
  1BFh+n*10h  3  chsFirst                                       ; four
  1C2h+n*10h  1  type (00h=unused, 01h=FAT12, 06h=FAT16B)       ; partitions
  1C3h+n*10h  3  chsLast                                        ; (n=0..3)
  1C6h+n*10h  4  lbaFirst  ;\logical block addresses/sizes      ;
  1CAh+n*10h  4  lbaSize   ;/counted in 200h-byte sectors       ;/
  1FEh        2  mbrsig (55h,AAh)                               ;-MBR ID
```

The CHS and LBA values are essentially containing the same information (CHS being an older standard, and LBA being invented in 1996). The 24bit CHS values are encoded as:

```
  0-7    Head Bit0-7     (00h..FEh) (or less common, 00h..FFh)
  8-13   Sector Bit0-5   (01h..3Fh)
  14-15  Cylinder Bit8-9
  16-23  Cylinder Bit0-7 (000h..3FFh, with above bit8-7)
```

To convert CHS to LBA, one must know the number of (logical) heads and sectors per cylinder (that info isn't stored in the MBR). The DSi's eMMC uses 32 sectors, 16 heads, 1024 cylinders:

```
  LBA = (Cylinder*32*16) + (Head*32) + (Sector-1)
```

Anyways, it's better/easier to use the LBA values directly, and ignore the CHS values.

3DS eMMC Partition table

Contents are unknown. The 3DS is said to have more than four partitions, so it must be using some extended MBR variant. Reportedly there's some extra "NCSD" header in the MBR sector.

SD/MMC Card Partition table

SD/MMC Cards are shipped with pre-formatted filesystem, so they should stick to some standarized MBR variant, with only a single partition used, possibly with additional date/id fields(?).
Note that SD/MMC Cards may or may not use partition tables (see CSD Register, FILE_FORMAT entry).

Formatting and Reformatting

SD/MMC cards are usually pre-formatted with empty filesystems. Reformatting should be avoided, or should be done only with tools that are aware of some memory card specific requirements:
The cluster size should not be less than the physical sector size (otherwise, when writing smaller clusters, sectors may get erased multiple times, which would result in slower writing and reduced lifetime).
Many devices do support only the standard FAT filesystems (for example, cameras or mp3-players may be unable to access memory cards with NTFS filesystem).

More info
For more info on MBR variants (and on the Partition Type value at 1C2h+n*10h), see:
```
http://en.wikipedia.org/wiki/Master_Boot_Record
http://en.wikipedia.org/wiki/Partition_type <-- rather meaningless
```

DSi SD/MMC Filesystem (FAT)

Volume Boot Record (VBR) (FAT12/FAT16) (DOS 4.0 EBPB)
```
000h 3   80x86 jump opcode       (DSi: E9h,00h,00h)
003h 8   ascii disk name         (DSi: "TWL     ")
00Bh 2   bytes / sector          (DSi: 0200h)
00Dh 1   sectors / cluster       (DSi: 20h)
00Eh 2   sectors / boot-record   (DSi: 0001h)
010h 1   number of FAT-copys     (DSi: 02h)
011h 2   entrys / root-directory (DSi: 0200h)
013h 2   sectors / disk          (DSi: 0000h)
015h 1   ID                      (DSi: F8h=HDD)
016h 2   sectors / FAT           (DSi: A:0034h, B:0009h)
018h 2   sectors / track         (DSi: 0020h)
01Ah 2   heads / disk            (DSi: 0010h)
01Ch 2   number of reserved sectors    (DSi: None such entry!)
01Ch 4   LBA First "hidden"      (DSi: A:00000877h, B:0006784Dh)
020h 4   LBA Size (total sectors)(DSi: A:00066F89h, B:000105B3h)
024h 1   Drive Number            (DSi: A:00h, B:01h)
025h 1   Flags (reserved)        (DSi: 00h)
026h 1   EBPB Version            (DSi: 29h) (that is, DOS 4.0 EBPB)
027h 4   Volume Serial Number    (DSi: 12345678h)
02Bh 11  Volume Label            (DSi: "           ")
036h 8   Filesystem Type         (DSi: 00h-filled)
03Eh 448 Bootcode                (DSi: 00h-filled)
1FEh 2   Signature               (DSi: 55h,AAh)
```
SDHC carts use FAT32, which differs in entries 011h, 016h and 024h-059h:
```
011h 2   Must be 0 (number of root entries, is variable-length FAT chain)
016h 2   Must be 0 (sectors per fat, instead use 32bit value at 024h)
024h 4   sectors / FAT  (new 32bit value instead of old entry 016h)
028h 2   ExtFlags (related to "active" FAT copy)
02Ah 2   Version of FAT32 filesystem (minor, major) (should be 0.0)
02Ch 4   Cluster number of first Root directory cluster (usually/often 2)
030h 2   Sector number of FSINFO in reserved area (usually 0001h)
032h 2   Sector number of VBR backup copy in reserved area (usually 0006h)
034h 12  Reserved for future   ;Should be zerofilled
040h 1   Drive Number                                          ;\
041h 1   Flags (reserved)                                      ; as old
042h 1   EBPB Version          ;Must be 29h (that is, DOS 4.0 EBPB) ; entries
043h 4   Volume Serial Number                                  ; at 024h
047h 11  Volume Label                                          ;
052h 8   Filesystem Type       ;Must be "FAT32   "             ;/
```
Moreover, FAT32 has the "FSINFO" at sector number VBR[30h]:
```
000h 4   Value 41615252h (aka "RRaA")
004h 480 Reserved (should be 0)
1E4h 4   Value 61417272h (aka "rrAa")
1E8h 4   Hint on number of free clusters   (or FFFFFFFFh=unknown)
1ECh 4   Hint on first free cluster number (or FFFFFFFFh=unknown)
```

```
  1F0h 12  Reserved (should be 0)
  1FCh 4   Value AA550000h
```
For more info see http://en.wikipedia.org/wiki/BIOS_Parameter_Block and fatgen103.pdf (official specs from microsoft).

File Allocation Table - FAT and FAT copy(s)
The following sectors are occupied by the File Allocation Table (FAT), which contains 12bit, 16bit, or 32bit entries (for FAT12/16/32) for each cluster:
```
  (x000)(0)000      unused, free
  (x000)(0)001      ???
  (x000)(0)002...   pointer to next cluster in chain (0)002..(F)FEF
  (xFFF)(F)FF0-6    reserved (no part of chain, not free)
  (xFFF)(F)FF7      defect cluster, don't use
  (xFFF)(F)FF8-F    last cluster of chain
```
The "x" in MSB of FAT32 entries is reserved (ie. FAT32 is restricted to 28bit cluster numbers).
Number and size of FATs can be calculated by the information in the boot sector.
Note on The first two FAT entries:
"The first cluster of the data area is cluster #2. That leaves the first two entries of the FAT unused. In the first byte of the first entry a copy of the media descriptor is stored. The remaining bits of this entry are 1. In the second entry the end-of-file marker is stored. The high order two bits of the second entry are sometimes, in the case of FAT16 and FAT32, used for dirty volume management: high order bit 1: last shutdown was clean; next highest bit 1: during the previous mount no disk I/O errors were detected.
(Historically this description has things backwards: DOS 1.0 did not have a BIOS Parameter Block, and the distinction between single-sided and double-sided 5.25" 360K floppies was indicated by the first byte in the FAT. DOS 2.0 introduced the BPB with media descriptor byte.)"

Root directory
The following sectors are the Root directory, again, size depends on the info in bootsector (on FAT32 is it's a normal cluster chain with variable size). Each entry consists of 32 bytes:
```
  00-07  8    Filename (first byte: 00=free entry, 2E=dir, E5=deleted entry)
  08-0A  3    Filename extension
  0B     1    Fileattribute
                bit0 read only
                bit1 hidden
                bit2 system
                bit3 volume label
                bit4 subdirectory
                bit5 archive-flag
                bit6 reserved
                bit7 reserved
  0C-0D  2    Reserved, or stuff
  0E-0F  2    Reserved, or Creation Timestamp
  10-11  2    Reserved, or Creation Datestamp
  12-13  2    Reserved, or Last Access Datestamp
  14-15  2    Reserved, or MSBs of Cluster (for FAT32)
  16-17  2    Last Modify Timestamp: HHHHHMMM, MMMSSSSS
  18-19  2    Last Modify Datestamp: YYYYYYYM, MMMDDDDD
  1A-1B  2    Pointer to first Cluster of file
  1C-1F  4    Filesize in bytes (always 0 for directories)
```
The 'cluster' entry points to the first used cluster of the file. The FAT entry for that cluster points to the next used cluster (if any), the FAT entry for that cluster points to the next cluster, and so on.

Long File Names (LFNs)
Long File Names (LFNs) are occupying multiple continous directory entries, consisting of a normal short filename entry, preceeded by one or more LFN prefix entries (with Attribute=0Fh). Each LFN prefix can hold 13 characters, the total length should not exceed 255 characters. The name should be terminated by 0000h, and any remaining unused characters should be padded by FFFFh. The LFN prefix entries are using the following format:
```
  00h  1  Sequence Number (bit6: last logical, first physical LFN entry,
```

```
         bit5: 0, bit4-0: number 01h..14h (1Fh)) (or E5h=deleted entry)
  01h  10 Long Filename characters (five UCS-2 characters)
  0Bh  1  Attributes (always 0Fh for LFN prefix)
  0Ch  1  Type (always 00h)
  0Dh  1  Short Filename Checksum
         sum=00h, for i=0 to 10, sum = (sum ROR 1) + shortname_char[i], next i
  0Eh  12 Long Filename characters (six UCS-2 characters)
  1Ah  2  First cluster (always 0000h)
  1Ch  4  Long Filename characters (two UCS-2 characters)
```

For example, "File with very long filename.ext" would be formatted as so:

```
  Entry 1: LFN Prefix (43h) "me.ext", 0000h, 6xFFFFh
  Entry 2: LFN Prefix (02h) "y long filena"
  Entry 3: LFN Prefix (01h) "File with ver"
  Entry 4: Normal 8.3 short filename entry "FILEWI~1.EXT"
```

http://en.wikipedia.org/wiki/Design_of_the_FAT_file_system#VFAT_long_file_names

Reserved Sectors (if any)

Usually the number of reserved sectors is zero. If it is non-zero, then the following sector(s) are reserved (and could be used by the boot procedure for whatever purposes).

Data Clusters 0002..nnnn

Finally all following sectors are data clusters. The first cluster is called cluster number (000)(0)002, followed by number (000)(0)003, (000)(0)004, and so on.

http://en.wikipedia.org/wiki/Design_of_the_FAT_file_system

DSi SD/MMC Internal NAND Layout

The DSi uses a 256MB Samsung eMMC moviNAND(?) flash chip, which is a NAND flash chip with a built-in controller that implements a MMC (SDIO?) interface. In many ways, it's like an SD card (or actually: MMC card) in BGA packaging, and some people have successfully read it with modified(why/how?) SD(/MMC?) card readers (is there any standard software that can be used for doing that?). The last 16MB is used for wear-leveling purposes (such as replacing bad blocks), while the first 240MB is used for storing actual data.

Addressing is done in terms of 512-bytes sectors. All wear-levelling and bad-block-mapping is handled transparently inside the chip by the controller. Most sectors are encrypted with a per-console key.

Overall eMMC Layout

```
  Offset     Size     Description
  00000000h  200h     PC-style MBR, encrypted with a per-console key
  00000200h  200h     Stage 2 Boot Info Block 1 (used)
  00000400h  200h     Stage 2 Boot Info Block 2 (unused, same as above)
  00000600h  200h     Stage 2 Boot Info Block 3 (unused, nonsense NAND offsets)
  00000800h  26600h   Stage 2 ARM9 Bootcode (encrypted with universal key)
  00026E00h  27600h   Stage 2 ARM7 Bootcode (encrypted with universal key)
  0004E400h  400h     Stage 2 Footer -- unknown format, but first 10 bytes
                        are (unencrypted) build number of Stage 2 bootloader
  0004E800h  B1000h   Unused (all 00h)
  000FF800h  200h     Unused (all 00h) (or No$gba Footer with CID & Console ID)
  000FFA00h  400h     Diagnostic area. (often contains build date of
                        device in plaintext) Blank in never-before-booted
                        DSi. Might be written to during firmware updates.
  000FFE00h  200h     Unused (all FFh)
  00100000h  EE00h    Unused (all 00h)
  0010EE00h  CDF1200h 1st partition (205.9Mbyte) (main, encrypted, FAT16)
  0CF00000h  9A00h    Unused (all 00h)
  0CF09A00h  20B6600h 2nd partition (32.7Mbyte)  (photo, encrypted, FAT12)
 For 240.0MB chips (Samsung KMAPF0000M-S998 or KLM5617EFW-B301):
```

```
  0EFC0000h  BA00h    Unused (all 00h)
  0EFCBA00h  34600h   3rd partition (0.2Mbyte)   (extra, unformatted)
  0F000000h  -        End of 240MByte Address Space
 For 245.5MB chips (ST NAND02GAH0LZC5, both rev30 and rev31):
  0EFC0000h  B600h    Unused (all 00h?) (smaller unused area as in 240MB chip)
  0EFCB600h  5B4A00h  3rd partition (5.7Mbyte)   (extra, unformatted)
  0F580000h  -        End of 245.5MByte Address Space
```

Stage 2 Boot Info Blocks 1, 2, 3 (unencrypted, aside from the RSA block)

```
  000h 20h Zerofilled
  020h 4   ARM9 Bootcode NAND Offset         (800h)      (Info Block 3: 80400h)
  024h 4   ARM9 Bootcode Size actual         (26410h)
  028h 4   ARM9 Bootcode RAM Address / Entry (37B8000h)
  02Ch 4   ARM9 Bootcode Size rounded-up     (26600h)
  030h 4   ARM7 Bootcode NAND Offset         (26E00h)    (Info Block 3: A6A00h)
  034h 4   ARM7 Bootcode Size actual         (27588h)
  038h 4   ARM7 Bootcode RAM Address / Entry (37B8000h)
  03Ch 4   ARM7 Bootcode Size rounded-up     (27600h)
  040h BFh Zerofilled
  0FFh 1   Unknown (0Ch)
  100h 80h RSA Block (B3,FF,EC,E5,..)       (Boot Info Block 3: 5B,E1,7A,9F,..)
  180h 14h Global MBK1..MBK5 Slot Settings
  194h 0Ch Local ARM9 MBK6..MBK8 Settings
  1A0h 0Ch Local ARM7 MBK6..MBK8 Settings
  1ACh 4   Global MBK9 Slot Write Protect (FF000000h)
  1B0h 50h Zerofilled
```

The above RSA Block contains 74h bytes of information (plus 0Bh bytes padding):

```
  Pre  0Bh  Leading RSA Padding (01,FF,FF,FF,FF,FF,FF,FF,FF,FF,00)
  00h  10h  AES_Engine Key Y for ARM9/ARM7 Bootcode (EC,07,00,00,...)
  10h  14h  SHA1 on WifiFlash[00h..27h] and eMMCBootInfo[00h..FFh,180h..1FFh]
            3DS: reportedly NAND/MBR[00h..27h] instead of WifiFlash[00h..27h]??
  24h  14h  SHA1 on decrypted ARM9 Bootcode, with the actual binary size
  38h  14h  SHA1 on decrypted ARM7 Bootcode, with the actual binary size
  4Ch  14h  Zerofilled
  60h  14h  SHA1 on above 60h-byte area at [00h..5Fh] (63,D2,FC,6E,...)
```

eMMC Encryption for Boot Sectors (AES-CTR, with fixed key; from RSA block)

The ARM9/ARM7 bootcode is encrypted via AES-CTR:

```
  RSA_KEY  = F1,F5,1A,FF,...         ;-from 3DS TWL_FIRM (for RSA Block)
  IV[0..3] = +size                   ;\
  IV[4..7] = -size                   ; size rounded up to 200h boundary, ie.
  IV[8..B] = -size-1                 ; from Boot Info Block entries [02Ch,03Ch]
  IV[C..F] = 00000000h               ;/
  KEY_X[0..F] = "Nintendo DS",...    ;-same as Key X for "Tad Files"
  KEY_Y[0..F] = EC,07,00,00,...      ;-from RSA Block (see above)
```

The RSA_KEY key is stored in some non-dumpable area of the DSi BIOS, making it impossible to obtain that key without chip decapping. However, Nintendo has included the same RSA_KEY in the "TWL_FIRM" firmware update for 3DS.

eMMC Encryption for MBR/Partitions (AES-CTR, with console-specific key)

The MBR and both partitions are encrypted via AES-CTR:

```
  IV[0..F]:    SHA1(CID)+Address/10h                    ;-eMMC Chip ID
  KEY_X[0..3]: [4004D00h]                               ;\
  KEY_X[4..7]: [4004D00h] XOR 24EE6906h                 ; CPU/Console ID, for
  KEY_X[8..B]: [4004D04h] XOR E65B601Dh                 ; DSi partitions on DSi
  KEY_X[C..F]: [4004D04h]                               ;/
  KEY_X[0..3]: [4004D00h]                               ;\CPU/Console ID, for
  KEY_X[4..B]: "NINTENDO"                               ; DSi partitions on 3DS
  KEY_X[C..F]: [4004D04h]                               ;/
  KEY_Y[0..F]: 0AB9DC76h,BD4DC4D3h,202DDD1Dh,E1A00005h  ;-Constant
```

The CID value (eMMC Chip ID) should be in same format as stored in RAM at 2FFD7BCh: little-endian 120bit (without crc7), padded to 128bit (with MSB=00h), ie. it should look like this (dd/ss being date/serial numbers):

```
CID = [2FFD7BCh] = dd,ss,ss,ss,ss,03,4D,30,30,46,50,41,00,00,15,00
SHA1(CID) = SWI_27h(SHA1value,2FFD7BCh,10h)
```
The resulting SHA1value is 14h-bytes, the first 10h-bytes are used as IV value, whereas the DSi doesn't adjust the endianness (it does just use the SWI's "big-endian" SHA1value as "little-endian" AES/IV value).
The CTR gets incremented after each 10h bytes (ie. to access a random address: "IV=SHA1value+(address/10h)".
For more info on obtaining the CID and Port [4004D00h] values, see:
DSi Console IDs
See also:
DSi SD/MMC Images

Related Decryption Tools
```
"NUS Downloader" allows to download and decrypt system updates
"DSi SRL Extract" allows to decrypt DSiware files (when copied to SD card)
"TWLTool" decrypt/encrypt eMMC images (firmware downgrading, dsiware-hax)
"TWLbf" and "bfCL" bruteforce Console ID or CID (or both) from eMMC images
```

DSi SD/MMC Bootloader

Stages
```
Stage 1: Load Stage 2 from NAND Boot Sectors (via code in BIOS ROM)
Stage 2: Load Stage 3 from NAND Filesystem
Stage 3: Contains GUI and allows to boot Cartridges or NAND files
```

Stage 1
The first stage of the DSi's bootloader lives in ROM, presumably on the CPU die. It loads further encrypted (and probably signed) stages from NAND flash, starting with a (partially unencrypted) offset table in the sector at 0x200.
Not much is known about this bootloader yet, but it presumably knows how to:
```
Initialize the encryption hardware
Read the contents of NVRAM
Initialize both LCDs
Read blocks (but not files) from the NAND flash
Perform some variety of integrity check on all data it reads(signature,CRC,?)
Display basic hexadecimal error codes
Possibly factory-programming the NAND flash?
Might also do basic power-on self test of peripherals
```
When the Stage 1 bootloader (in ROM) fails, it displays a 32-bit hexadecimal number on the top screen, known Stage 1 error codes are:
```
Error Code Description
0000FE00   Error communicating NAND chip (It's missing, CLK shorted, etc.)
0000FEFC   Integrity error in first block of Stage 2 (address at 220h)
0000FEFD   Integrity error in second block of Stage 2 (address at 230h)
0000FEFE   Boot sector integrity error (Sector 200h not valid), or error
             in NVRAM contents.
```

Stage 2
Unlike the stage1 bootloader, which must be small enough to fit in ROM (probably several kilobytes), the stage2 bootloader has about a megabyte of NAND flash reserved for it. The stage2 bootloader understands partitions and filesystems, and it is capable of loading the DSi menu. It also must understand the encryption used on filesystem blocks in the NAND, and it must understand how to load and validate title metadata.
The Stage 2 loader was not modified by the System Menu 1.4 update. This is still earlier in the boot process than the "Health and Safety" warning.
The first stage bootloader reads sector 0x200 in order to find a table of offsets to the Stage 2 bootloader:
```
00000220  00 08 00 00 10 64 02 00  00 80 7b 03 00 66 02 00 |.....d....{..f..|
00000230  00 6e 02 00 88 75 02 00  00 80 7b 03 00 76 02 00 |.n...u....{..v..|
00000240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
```

This appears to be describing two chunks of the stage2 loader, one 0x26410 bytes in length at address 0x800, and one 0x27588 bytes at address 0x26e00.
Note that this sector (and two similar ones at 0x400 and 0x600) appear to be the only unencrypted blocks on the NAND flash.
It is unclear why there are two pieces which are nearly but not quite the same size. Passive traces of the boot sequence confirm that the 0x26e00 chunk is slightly larger, and it's loaded first. The 0x800 chunk is read immediately after the 0x26e00 chunk.
Whereas the filesystem data in NAND is encrypted using a unique key for every DSi, the stage2 bootloader is identical on every DSi tested so far. This probably means that it is encrypted using a fixed key included in stage1.

After Stage 2 is loaded:

1. The NAND flash is partially re-initialized
2. Sector 0 is read from the NAND. Appears to be (encrypted) DOS-style MBR.
3. The MBR signature and the type of the first partition are verified.
4. Filesystem metadata is read from sectors starting around 0x100000. The metadata appears to be in FAT format with long filenames.
5. Multiple files are loaded from the filesystem. The exact read addresses will vary depending on your DSi's firmware version and the state of its filesystem when you performed the last firmware update. On a brand new DSi, it appears that the DSi Menu itself is loaded from 0xb20000 after two small metadata files are read from 0xb1c000 and 0x7a0000.

All Stage 2 errors show before the health and safety screen. It appears that Stage 2 errors from a cold power-on always cause the DSi to hang at a black screen, whereas Stage 2 errors after reset (pressing but not holding the power button) will give an error message screen. Known Stage 2 errors:

Text	Description
"Error: 1-2435-8325"	Invalid signature or partition type in MBR, invalid starting LBA.
"Error: 2-2435-8325"	Error reading fat/sectors from eMMC
"Error: 3-2435-8325"	DSi Menu integrity checks failed

Boot Sectors in DSi Debug version
Debug version bootsectors are almost same as in retail version: There's an extra RSA key inserted at ARM9:37CEADCh, and lots of B/BL/BLX/LDR addresses have changed due to the inserted bytes, but, apart from that changes, only 5 opcodes are really different, ARM9:37C07A0h, ARM9:37C0864h, ARM7:37B9110h, ARM7:37B9200h do trigger errors when SCFG_DEBUG bit0-1 are zero (opposite of retail version), and ARM9:37B8BA4h uses the extra RSA key at 37CEADCh for Debug Launcher (instead of the retail key at 1FFC400h).

DSi SD/MMC Device List

The Device List is automatically copied to ARM7 RAM address defined in cartheader[1D4h] by firmware. The 400h-byte list is mainly used for DSiware/firmware titles (DSi ROM titles will usually just receive an empty zerofilled 400h-byte list; unless maybe if they are flagged to use SD/MMC hardware in cartheader?). There appears to be no range check for the cartheader[1D4h] entry (setting it to zero causes the list to be 'written' to ARM7 ROM area).

Device List, 400h-bytes, loaded to ARM7/RAM address from cartheader[1D4h]

000h	54h*11	Device List (max 11 entries)
39Ch	24h	Zerofilled
3C0h	40h	Name 'nand:/title/000300tt/4ggggggg/content/000000vv.app' + 00h's

Format of the 54h-byte device list entries:

00h	1	Drive Letter ("A".."I")
01h	1	Flags (see below)
02h	1	Access Rights (bit1=Write, bit2=Read)

```
  03h  1    Zero
  04h  10h  Device Name (eg. "nand" or "dataPub") (zeropadded)
  14h  40h  Path        (eg. "/" or "nand:/shared1") (zeropadded)
```
Bits in Flags byte:
```
  0    Physical Drive (0=External SD/MMC Slot, 1=Internal eMMC)
  1-2  Zero (maybe MSBs of Drive)
  3-4  Device Type (0=Physical, 1=Virtual/File, 2=Virtual/Folder, 3=Reserved)
  5    Partition (0=1st, 1=2nd)
  6    Zero (maybe MSB of Partition)
  7    Encrypt? (set for eMMC physical devices; not for virtual, not for SD)
```
The DSi has 9 default devices ("A"-"I"):
```
  Letter/Flags     Name       Path                    ;Notes
  'A',81h,06h,00h  'nand'     '/'                     ;eMMC Cart Partition 1
  'B',A1h,06h,00h  'nand2'    '/'                     ;eMMC Cart Partition 2
  'C',11h,04h,00h  'content'  'nand:/title/000300tt/4gggggggg/content'
  'D',11h,04h,00h  'shared1'  'nand:/shared1'         ;TWLCFGn.dat
  'E',11h,06h,00h  'shared2'  'nand:/shared2'         ;Sound and wrap.bin
  'F',31h,06h,00h  'photo'    'nand2:/photo'          ;Camera photos/frames
  'G',09h,06h,00h  'dataPrv'  'nand:/title/000300tt/4gggggggg/data/private.sav'
  'H',09h,06h,00h  'dataPub'  'nand:/title/000300tt/4gggggggg/data/public.sav'
  'I',00h,06h,00h  'sdmc'     '/'                     ;SD Cart Partition 1
```
Depending on the cartheader, usually only 5-8 of that 9 devices are copied to the device list in RAM, and Access Rights for some devices can be crippled in the RAM list entries).
Caution: The list may not contain forward references (eg. one can redirect "dataPub" to "sdmc:/flipnote.pub", but that works only if "sdmc" was already defined in one of the previous entries).
A and B: These physical eMMC partitions are always included in the list (because they are needed as parent entries for Virtual devices), however, their Access Rights are usually set to 00h (unless Title ID indicates a system file; with cartheader[234h].bit4=1).
C and E: These virtual devices are passed ONLY to System Menu (Launcher)... hmmm, or actually, the Launcher doesn't receive ANY device list at all?
G and H: Present only if Title ID indicates DSiware/firmware; with cartheader[234h].bit2=1 (ie. not ROM carts), and only if public/private.sav sizes are nonzero in carthdr[238h/23Ch].
A-H: removed from list if carthdr[1B4h].4=0 (no eMMC access).
I: removed from list if carthdr[1B4h].3=0 (no SD card access).
Some games may adjust the Device List AFTER booting. For example, Flipnote changes the "photo" (F) Access Rights to 04h, and appends an extra device at the end of the list, using a spare drive letter (C):
```
  'C',09h,06h,00h  'share'    'nand:/shared2/0000'   ;Sound file
```
Other device names that have been spotted here or there:
```
  'verdata'   for Version Data NARC file
  'rom'       for executable's NitroROM filesystem
  'otherPub'
  'otherPrv'
```
Apart from Virtual Devices, there are also some Virtual Filename Placeholders:
```
  'nand:/<tmpjump>'     --> 'nand:/tmp/jump.app'
  'nand:/<sharedFont>'  --> 'nand:/sys/TWLFontTable.dat'
  'nand:/<verdata>'     --> 'nand:/title/0003000f/484e4c%02x/content/%08x.app'
  'nand:/<banner>'      --> ..... '/data/banner.sav'
  ':<srl>'              --> .....
```

public & private savedata

If the file is in a folder named "CONTENT", then savedata is stored in a separata "DATA" folder (this is as how Nintendo is doing it officially):
```
  "nand:/../CONTENT/title.tmd"
  "nand:/../CONTENT/00000002.app"
  "nand:/../DATA/public.sav"
  "nand:/../DATA/private.sav"
```
If the file is anywhere else (not in a CONTENT folder), then savedata should be stored in the same folder & same name, with extension changed to .pub or .prv (this homebrew convention allows to use more descriptive non-numeric filenames, and doesn't require "title.tmd"):
```
  "sdmc:/../My Folder Name/Filpnote Studio (EUR.AUS).dsi"
```

```
  "sdmc:/../My Folder Name/Filpnote Studio (EUR.AUS).pub"
  "sdmc:/../My Folder Name/Filpnote Studio (EUR.AUS).prv"
```
As seen above, names can exceed the 8.3 character shortfilename limit (and can use 16bit unicode), however, using such long filenames can quickly exceed the 64 character limit of the DSi's Device List (or it's 8bit character space), as a workaround, unlaunch is converting the above names to 8.3 character format before storing them in the Device List:
```
  "sdmc:/../MYFOLD~1/FLIPN~12.DSI"
  "sdmc:/../MYFOLD~1/FLIPN~10.PUB"
  "sdmc:/../MYFOLD~1/FLIPNO~2.PRV"
```
With those short 8-character folder names, one can have up to five folders nested (or only three folders with extension alike "MYFOLD~1.EXT", or more than five folders if folder/filename are shorter than 8 characters). As shown in the above examples, the longname ("Filpnote Studio (EUR.AUS)") is same for all three files, but the ~NN suffix may vary for shortnames.

DSi SD/MMC Complete List of SD/MMC Files/Folders

DSi eMMC Partition 1 (FAT16)
```
  SYS          <DIR>    sys
  LOG          <DIR>     log
  PRODUCT  LOG 0000023D   product.log
  SYSMENU  LOG 00004000   sysmenu.log
  SHOP     LOG 00000020   shop.log
  HWINFO_S DAT 00004000  HWINFO_S.dat
  HWINFO_N DAT 00004000  HWINFO_N.dat
  CERT     SYS 00000F40  cert.sys
  HWID     SGN 00000100  HWID.sgn
  TWLFON~1 DAT 000D2C40  TWLFontTable.dat
  DEV      KP  000001BE  dev.kp
  TITLE        <DIR>    title
  00030017     <DIR>     00030017 (aka System Menu)
  484E4150     <DIR>      484e4150 (aka Launcher)
  DATA         <DIR>       data
  PRIVATE  SAV 00004000     private.sav
  CONTENT      <DIR>       content
  TITLE    TMD 00000208     title.tmd
  00000002 APP 0019E400     00000002.app
  00030015     <DIR>     00030015 (aka System Base Tools)
  484E4250     <DIR>      484e4250 (aka System Settings)
  DATA         <DIR>       data
  CONTENT      <DIR>       content
  TITLE    TMD 00000208     title.tmd
  00000002 APP 00285C00     00000002.app
  484E4650     <DIR>      484e4650 (aka Nintendo DSi Shop)
  DATA         <DIR>       data
  PRIVATE  SAV 00004000     private.sav
  EC       CFG 00000134     ec.cfg
  CONTENT      <DIR>       content
  00000004 APP 00526400     00000004.app
  TITLE    TMD 00000208     title.tmd
  0003000F     <DIR>     0003000f (aka System Data)
  484E4341     <DIR>      484e4341 (aka Wifi Firmware)
  DATA         <DIR>       data
  CONTENT      <DIR>       content
  TITLE    TMD 00000208     title.tmd
  00000002 APP 00017E60     00000002.app
  484E4841     <DIR>      484e4841 (aka Nintendo DS Cart Whitelist)
  DATA         <DIR>       data
  CONTENT      <DIR>       content
  TITLE    TMD 00000208     title.tmd
  00000001 APP 0004B1D0     00000001.app
  484E4C50     <DIR>      484e4c50 (aka Version Data)
```

```
DATA         <DIR>       data
CONTENT      <DIR>       content
00000004 APP 00001B50     00000004.app
TITLE    TMD 00000208     title.tmd
00030005     <DIR>     00030005 (aka System Fun Tools)
484E4441     <DIR>      484e4441 (aka DS Download Play)
DATA         <DIR>       data
CONTENT      <DIR>       content
TITLE    TMD 00000208     title.tmd
00000001 APP 00069BC0     00000001.app
484E4541     <DIR>      484e4541 (aka Pictochat)
DATA         <DIR>       data
CONTENT      <DIR>       content
00000000 APP 00074FC0     00000000.app
TITLE    TMD 00000208     title.tmd
484E4950     <DIR>      484e4950 (aka Nintendo DSi Camera)
DATA         <DIR>       data
PRIVATE  SAV 00080000     private.sav
CONTENT      <DIR>       content
TITLE    TMD 00000208     title.tmd
00000002 APP 00443C00     00000002.app
484E4A50     <DIR>      484e4a50 (aka Nintendo Zone)
DATA         <DIR>       data
PRIVATE  SAV 00100000     private.sav
CONTENT      <DIR>       content
00000003 APP 0014D000     00000003.app
TITLE    TMD 00000208     title.tmd
484E4B50     <DIR>      484e4b50 (aka Nintendo DSi Sound)
DATA         <DIR>       data
PRIVATE  SAV 00080000     private.sav
CONTENT      <DIR>       content
00000002 APP 00451000     00000002.app
TITLE    TMD 00000208     title.tmd
00030004     <DIR>     00030004 (aka DSiware)
484E4750     <DIR>      484e4750 (aka Nintendo DSi Browser)
DATA         <DIR>       data
PRIVATE  SAV 00200000     private.sav
CONTENT      <DIR>       content
00000001 APP 008F1C00     00000001.app
TITLE    TMD 00000208     title.tmd
4B475556     <DIR>      4b475556 (aka Flipnote Studio)
DATA         <DIR>       data
PUBLIC   SAV 007F0000     public.sav
CONTENT      <DIR>       content
00000000 APP 00348400     00000000.app
TITLE    TMD 00000208     title.tmd
TICKET       <DIR>    ticket
00030017     <DIR>     00030017 (aka System Menu)
484E4150 TIK 000002C4   484e4150.tik (aka Launcher)
00030015     <DIR>     00030015 (aka System Base Tools)
484E4250 TIK 000002C4   484e4250.tik (aka System Settings)
484E4650 TIK 000002C4   484e4650.tik (aka Nintendo DSi Shop)
0003000F     <DIR>     0003000f (aka System Data)
484E4341 TIK 000002C4   484e4341.tik (aka Wifi Firmware)
484E4841 TIK 000002C4   484e4841.tik (aka Nintendo DS Cart Whitelist)
484E4C50 TIK 000002C4   484e4c50.tik (aka Version Data)
00030005     <DIR>     00030005 (aka System Fun Tools)
484E4441 TIK 000002C4   484e4441.tik (aka DS Download Play)
484E4541 TIK 000002C4   484e4541.tik (aka Pictochat)
484E4950 TIK 000002C4   484e4950.tik (aka Nintendo DSi Camera)
484E4A50 TIK 000002C4   484e4a50.tik (aka Nintendo Zone)
484E4B50 TIK 000002C4   484e4b50.tik (aka Nintendo DSi Sound)
00030004     <DIR>     00030004 (aka DSiware)
484E4750 TIK 000002C4   484e4750.tik (aka Nintendo DSi Browser)
4B414D56 TIK 000002C4   4b414d56.tik (aka Paper Plane)
```

```
4B443956 TIK 000002C4   4b443956.tik (aka Dr. Mario)
4B475556 TIK 000002C4   4b475556.tik (aka Flipnote Studio)
4B4D3958 TIK 000002C4   4b4d3958.tik (aka Magic Made Fun: Deep Psyche)
SHARED1      <DIR>    shared1
TWLCFG0  DAT 00004000  TWLCFG0.dat
TWLCFG1  DAT 00004000  TWLCFG1.dat
SHARED2      <DIR>    shared2
LAUNCHER     <DIR>     launcher
WRAP     BIN 00004000   wrap.bin
0000         00200000  0000
IMPORT       <DIR>    import
TMP          <DIR>    tmp
ES           <DIR>     es
WRITE        <DIR>      write
PROGRESS     <DIR>    progress
```

DSi eMMC Partition 2 (FAT12)

```
PHOTO        <DIR>    photo
PRIVATE      <DIR>     private
DS           <DIR>      ds
APP          <DIR>       app
484E494A     <DIR>        484E494A (aka Nintendo DSi Camera Stuff)
PIT      BIN 00001F60      pit.bin
DCIM         <DIR>     DCIM
100NIN02     <DIR>      100NIN02
HNI_0008 JPG 0000AB51    HNI_0008.JPG
HNI_0009 JPG 00009A96    HNI_0009.JPG
HNI_0010 JPG 0000932B    HNI_0010.JPG
HNI_0011 JPG 00009CB8    HNI_0011.JPG
HNI_0012 JPG 00009CA9    HNI_0012.JPG
HNI_0013 JPG 00009A3B    HNI_0013.JPG
```

DSi eMMC Partition 3 (unformatted)

There's a small 3rd partition in MBR, but it's left unformatted (the VBR and FAT and everything is left zero-filled). Unknown if there are any cases where this partition is used, and if so: if it's meant to be encrypted or unencrypted. In case it's meant to be encrypted: Observe that the unformatted partition contains UNENCRYPTED zeroes (so trying to "decrypt" those zeroes would produce random garbage data).

SD Card

DSiware (including Browser and Flipnote) can be exported to SD Card (via System Menu, Data Managment). Camera Photos and Frames can be exported (via Nintendo DSi Camera, Options, Copy) (Frames are some sort of gaudi-masks that can be used (and created) via one of the Camera special effect features; the mask uses YELLOW as transparent color). As on eMMC storage, all photos from internal camera are mirrored horizontally.

```
PRIVATE      <DIR>    private
DS           <DIR>     ds
TITLE        <DIR>      title                                    ;\
484E4750 BIN 9.180K      484E4750.bin (aka Nintendo DSi Browser)  ; dsiware
4B475556 BIN 11.510K     4B475556.bin (aka Flipnote Studio)       ; games
HNB_     LST 2K          HNB_.lst     (content: "VUGKPGNH")       ;/
APP          <DIR>      app
484E494A     <DIR>       484E494A (aka Nintendo DSi Camera Stuff) ;\
PIT      BIN 47K          pit.bin                                  ; camera
DCIM         <DIR>        DCIM                                     ; frames
100NIN02     <DIR>         100NIN02                                ;
HNI_0001 JPG 45K            HNI_0001.JPG  ;-frame/mask            ;/
4B475556     <DIR>       4B475556 (aka Flipnote Studio Stuff)     ;\
RECENT10 PLS 4K           recent10.pls                             ;
MARK0    PLS 8K           mark0.pls                                ; flipnote
MARK1    PLS 8K           mark1.pls                                ; stuff
MARK2    PLS 8K           mark2.pls                                ;
MARK3    PLS 8K           mark3.pls                                ;
```

```
001          <DIR>        001                                    ;
DIRMEMO2 LST 157K         dirmemo2.lst                           ;
F08243~1 PPM 467K         F08243_0E5E2296197E5_000.ppm           ;/
DCIM         <DIR>    DCIM
101NIN02     <DIR>     101NIN02          ;<-- can be 100NIN02 thru 999NIN02
HNI_0001 JPG 43K        HNI_0001.JPG      ;\dsi camera photos
HNI_0002 JPG 17K        HNI_0002.JPG      ; (names are numbered differently
HNI_0003 JPG 39K        HNI_0003.JPG      ;/as on eMMC where they came from)
```

Reportedly, the "Nintendo DSi Sound" application can also access audio files in AAC format, saved in any folders on SD Card.
Aside from DSi-related files, the SD Card can also contain whatever files from other computers.

Blocks
The System Menu, Data Management feature is referring to filesizes & free space in "Blocks" (aka 128Kbytes units, aka 1Mbit units).

DSi SD/MMC Summary of SD/MMC Files/Folders

File/folder names
The DSi is using weird numeric strings as file/folder names:

```
000000vv Title Version (lowercase hex32bit) from tmd[1E4h] as carthdr[1Eh]
4ggggggg Title ID Gamecode (hex) as carthdr[230h..233h]
000300tt Title ID Filetype (hex) as carthdr[234h..237h]
HNI_nnnn Camera photo/frame files (nnnn = 0001..0100 decimal)
nnnNIN02 Camera photo/frame folders (nnn = 100..999 decimal)
```

The "000300tt" can be:

```
00030000 ROM Cartridges (as so for ROMs, doesn't appear in SD/MMC files)
00030004 DSiware (browser, flipnote, and games) (if any installed)
00030005 System Fun Tools (camera, sound, zone, pictochat, ds download play)
0003000f System Data (non-executable, without carthdr)
00030015 System Base Tools (system settings, dsi shop, 3ds transfer tool)
00030017 System Menu (launcher)
```

The "4ggggggg" can be (last two digits are region(s), or "41" for all regions):

```
484e41gg System Menu (Launcher)
484e42gg System Settings
484e4341 Wifi Firmware (non-executable datafile) (all regions)
484e4441 DS Download Play (all regions)
484e4541 Pictochat (all regions) (no update available)
484e46gg Nintendo DSi Shop
484e47gg Nintendo DSi Browser
484e4841 Nintendo DS Cart Whitelist (non-executable datafile) (all regions)
484e49gg Nintendo DSi Camera
484e4agg Nintendo Zone (doesn't exist in Korea)
484e4bgg Nintendo DSi Sound
484e4cgg Version Data (non-executable datafile)
484e4fgg Nintendo 3DS Transfer Tool
484E494A Nintendo DSi Camera Data (uppercase) ("japan") (aka all regions)
4b44474a Dokodemo Wii no Ma (japan only)
4b4755gg Flipnote Studio (doesn't exist in Korea/China)
42383841 DS Internet settings (a new DSi tool on 3DS consoles)
4bgggggg DSiware games... (whatever games you have purchased, if any)
```

These files can be stored in Internal eMMC, or on external SD card, and can be downloaded from Nintendo's server (when buying games, or updating system files).

DSi Internal eMMC
Internal eMMC can contain System files and any purchased DSiware games:

```
FAT16:\ticket\000300tt\4ggggggg.tik (encrypted)      ;ticket (708 bytes)
FAT16:\title\000300tt\4ggggggg\content\title.tmd     ;tmd (520 bytes)
FAT16:\title\000300tt\4ggggggg\content\000000vv.app  ;executable (decrypted)
```

```
FAT16:\title\000300tt\4gggggg\data\public.sav       ;size as carthdr[238h]
FAT16:\title\000300tt\4gggggg\data\private.sav      ;size as carthdr[23Ch]
FAT16:\title\000300tt\4gggggg\data\ec.cfg           ;dsi shop only
FAT16:\title\000300tt\4gggggg\data\banner.sav       ;if carthdr[1BFh].bit2=1
```

Note that some of the above files are containing their own virtual filesystem inside of the eMMC's FAT16 filesystem (NitroROM filesystems in "000000vv.app" files, and FAT12 filesystems in "public.sav" and "private.sav" files).

The System tools (menu, settings, and shop) are also storing further data on FAT16 (outside of the ticket and title folders):

```
FAT16:\shared1\TWLCFG0.dat           ;16K
FAT16:\shared1\TWLCFG1.dat           ;16K
FAT16:\shared2\launcher\wrap.bin     ;16K
FAT16:\shared2\0000                  ;2048K (sound recorder)
FAT16:\sys\log\product.log           ;573 bytes
FAT16:\sys\log\sysmenu.log           ;16K
FAT16:\sys\log\shop.log              ;32 bytes
FAT16:\sys\HWINFO_S.dat              ;16K
FAT16:\sys\HWINFO_N.dat              ;16K
FAT16:\sys\cert.sys                  ;3904 bytes (or 2560 bytes)
FAT16:\sys\HWID.sgn                  ;256 bytes (unknown purpose/content)
FAT16:\sys\TWLFontTable.dat          ;843.1K (D2C40h bytes) (compressed)
FAT16:\sys\dev.kp                    ;446 bytes (encrypted)
FAT16:\import\                       ;empty folder
FAT16:\progress\                     ;empty folder
FAT16:\tmp\es\write\                 ;empty folder
```

The Camera is storing further data on the eMMC FAT12 partition:

```
FAT12:\photo\DCIM\100NIN02\HNI_nnnn.JPG                      ;camera photos
FAT12:\photo\private\ds\app\484E494A\pit.bin                  ;camera info
FAT12:\photo\private\ds\app\484E494A\DCIM\100NIN02\HNI_nnnn.JPG;camera frames
```

And, there's a small 3rd eMMC partition in MBR, but it's left unformatted (the VBR and FAT and everything is left zero-filled).

DSi External SD Card

DSiware games (and browser and flipnote) can be copied to SD card (via System Menu, Data Managment) (however, the DSi doesn't seem to allow to execute files on SD card, so they can be used only if they are copied back to the DSi):

```
SD:\private\ds\title\4GGGGGGG.bin    ;executable/data in one file (encrypted)
SD:\private\ds\title\HNB_.lst        ;list of gamecodes
```

Camera data can be copied to SD card (via Nintendo DSi Camera, Options, Copy):

```
SD:\DCIM\nnnNIN02\HNI_nnnn.JPG                          ;camera photos
SD:\private\ds\app\484E494A\pit.bin                     ;camera info
SD:\private\ds\app\484E494A\DCIM\nnnNIN02\HNI_nnnn.JPG   ;camera frames
```

Flipnote "movies" can be also saved on SD card:

```
SD:\private\ds\app\4B4755GG\recent10.pls     ;Recently saved path/filenames
SD:\private\ds\app\4B4755GG\mark0.pls        ;Heart sticker path/filenames
SD:\private\ds\app\4B4755GG\mark1.pls        ;Crown sticker path/filenames
SD:\private\ds\app\4B4755GG\mark2.pls        ;Music sticker path/filenames
SD:\private\ds\app\4B4755GG\mark3.pls        ;Skull sticker path/filenames
SD:\private\ds\app\4B4755GG\001\dirmemo2.lst ;List of all files in folder
SD:\private\ds\app\4B4755GG\001\XNNNNN_NNNNNNNNNNNNN_NNN.ppm          ;normal
SD:\private\ds\app\4B4755GG\YYYYMMDD\NNN\XNNNNN_NNNNNNNNNNNNN_NNN.ppm ;backup
SD:\private\ds\app\4B4755GG\gif\XNNNNN_NNNNNNNNNNNNN_NNN.gif          ;gif
```

The Nintendo DSi Sound utility can read .AAC (and .M4A) files from SD card (though it doesn't seem to allow to save your own recordings to SD card?). There appears to be no special folder location, ie. the AAC/M4A files can be anywhere:

```
SD:\...\*.aac
SD:\...\*.m4a
```

DSi Shop and System Update Download URLs

```
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4gggggg/tmd
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4gggggg/cetk
```

```
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4ggggggg/000000vv
```
The "cetk" file contains the encrypted ticket. The "cetk" file is available only for freeware downloads (eg. system updates). Commercial DSi Shop titles can be downloaded the same way, except that the ticket must be somehow purchased/downloaded separetely.
Nintendo does keep hosting older "00000vv" versions (except, the original version, "00000000" isn't available in all cases; namely if the title was pre-installed on all DSi's then it would be pointless to provide it as update).

NUS Downloader Notation
The homebrew NUS Downloader utility for PCs comes up with its own renaming scheme:
```
d:\...\TITLES\000300tt4ggggggg\ddd\000000vv      ;executable (encrypted)
d:\...\TITLES\000300tt4ggggggg\ddd\000000vv.APP  ;executable (decrypted)
d:\...\TITLES\000300tt4ggggggg\ddd\CETK          ;cetk (2468 bytes)
d:\...\TITLES\000300tt4ggggggg\ddd\TMD           ;tmd (520 bytes)
```
Whereas, "ddd" is same as "000000vv" multiplied by 256 decimal (which is nonsense and redundant). The decrypted ".APP" file is generated only if the "common key" is found in 16-byte file "dsikey.bin".

DSi SD/MMC Images

Filesystem Viewer
no$gba debug version is allowing to view the filesystem tree from encrypted eMMC images (via menubar: Window, Filesystem), the filesystem viewer is also able to export single files from eMMC and SD images (by double-clicking separate files).
```
XXX    currently, the double-clicked file will be saved as "TEMP.TMP",
XXX    in no$gba folder (there is no "Save as" dialog yet)
```

Emulation
No$gba emulates read/write-accesses to eMMC images.
Another idea for future would be using files & folders on the PC filesystem instead of a single image file (that might be easier to deal with in some cases, but for the reverse-engineering stage it's better to stick with original authentic images).

Encrypted eMMC Images
no$gba can emulate up to 12 consoles simultaneously, and expects the eMMC images having following filename(s), in no$gba folder:
```
DSi-#.mmc             ;eMMC for machine 1..12 (# = 1..C hex)
```
the eMMC images are encrypted with per-console keys, so decryption won't work without knowing the console ID values. no$gba expects that info to be stored in a 40h-byte Footer at the end of the file:
```
00000000h .. Encrypted eMMC image (usually 240Mbyte for DSi)
0F000000h 16 Footer ID      ("DSi eMMC CID/CPU")
0F000010h 16 eMMC CID       (dd ss ss ss ss 03 4D 30 30 46 50 41 00 00 15 00)
0F000020h 8  CPU/Console ID (nn n1 nn nn nn nn xn 08)
0F000028h 24 Reserved       (zerofilled)
```
Alternately, the "footer" can be stored in the zerofilled area at eMMC offset 000FF800h..000FF83Fh (using that area, the data can be kept in place even when using other tools; that were getting confused by the data appended at end of file).

SD Card Images
no$gba supports SD Card images in similar fashion as above eMMC images, but without needing any footer since there's encryption on SD cards. The image should contain a pre-formatted MBR and FAT (as real SD cards do).
```
DSi-#.sd              ;SD Card for machine 1..12 (# = 1..C hex)
```
note: no$gba does currently support only 128MB SDSC images (the CID, CSD, OCR, SCR, SSR registers are hardcoded for images with 125698048 byte size), there is a .zip file with an empty pre-formatted SD image in the no$gba package (if you want to use that image: unzip it to the no$gba folder).

Dumping eMMC Images
DSiware exploits like sudokuhax are allowing to access SD/MMC hardware (so one could simply copy all eMMC sectors to a file on SD Card).
Unfortunately, most/other exploits don't have SD/MMC access, so dumping would work only when the eMMC chip to a SD/MMC card reader. For details, see:
AUX DSi SD/MMC Pin-Outs
For obtaining the Console IDs, see:
DSi Console IDs

DSi SD/MMC DSiware Files on Internal eMMC Storage

DSiware games (downloaded from DSi Shop), and pre-installed System Tools are consisting of following files:

FAT16:\title\000300tt\4ggggggg\content\000000vv.app ;executable (decrypted)
Contains the executable, with same header as used in Cartridge ROM images, and usually with a NitroROM File System (ie. a second virtual filesystem inside of the FAT16 filesystem).
DSi Cartridge Header
DS Cartridge NitroROM and NitroARC File Systems
Note: There are also three non-executable ".app" files without cartridge headers (Wifi Firmware, Version Data, and DS Cart Whitelist).

FAT16:\title\000300tt\4ggggggg\data\public.sav ;size as carthdr[238h]
FAT16:\title\000300tt\4ggggggg\data\private.sav ;size as carthdr[23Ch]
These files can contain whatever save data. The .sav files are usually containing a FAT12 with its own VBR, FAT, and Directories (so they use some virtual FAT12 inside of the real FAT16).
When exporting a game to SD Card (via System Settings, Data Managment), then public.sav (eg. used by Flipnote) will be included in the exported image, whilst private.sav (eg. used by DSi Browser) won't be included.

FAT16:\title\000300tt\4ggggggg\data\ec.cfg ;dsi shop only
Whatever extra file, encrypted, 134h bytes, used by DSi Shop only.

FAT16:\title\000300tt\4ggggggg\data\banner.sav ;if carthdr[1BFh].bit2=1
Custom icon, used by some games to indicate the game progress. Format is similar as Icon/Title, but containing only animated icon data (without title). For details, see:
DS Cartridge Icon/Title

FAT16:\title\000300tt\4ggggggg\content\title.tmd ;tmd (520 bytes)
FAT16:\ticket\000300tt\4ggggggg.tik (encrypted) ;ticket (708 bytes)
FAT16:\ticket\000300tt\00000000.tik (encrypted) ;multi-tik? (N*708 bytes?)
These files do contain title metadata (.tmd) and tickets (.tik).
DSi SD/MMC DSiware Tickets and Title metadata
The .tik files are encrypted with ES Block Encryption (using same key X/Y as for dev.kp):

```
KEY_X[00h..03h] = 4E00004Ah                          ;\
KEY_X[04h..07h] = 4A00004Eh                          ; same as for Tad
KEY_X[08h..0Bh] = Port[4004D00h+4] xor C80C4B72h     ;
KEY_X[0Ch..0Fh] = Port[4004D00h+0]                   ;/
KEY_Y[00h..0Fh] = Constant (E5,CC,5A,8B,...)  ;from ARM7BIOS
```

DSi ES Block Encryption
Caution: There are some ways to modify .tmd files, but that can cause the whole title to be deleted when starting one of the following three tools:

```
Data Management (in System Settings), DSi Shop, and 3DS transfer tool
```

These tools will delete the titles "content" folder (with .app and .tmd files) and the "data" folder (with .sav files).
As a workaroung: Set read-only attribute for .tmd and .app files (the deletion aborts once when hitting a read-only file; with the files being processed as ordered in the directory).
Note: Tickets are kept stored in eMMC even after deleting titles (that's allowing to redownload the titles for free; at least that's been the case when the DSi shop was still online).

Below "wrap.bin" and "menusave.dat" files are containing lists of installed titles, however, it isn't neccessary to edit those files when manually installing .tmd/.app/.tik files.

FAT16:\shared2\launcher\wrap.bin (16Kbytes)
Contains a list of installed DSiware Title IDs (in no specific order).

```
0000h 14h   SHA1 on entries [014h..03Fh]
0014h 14h   SHA1 on entries [040h..177h]
0028h 4     ID ("APWR") (aka 'WRAP' with mis-ordered letters)
002Ch 4     Size of entries at [040h..177h] (00000138h, aka 39*8)
0030h 10h   Zerofilled
0040h 138h  Space for 39 Title IDs (as at cart[230h]) (8x00h=unused entry)
0178h 3E88h Unknown (looks like random/garbage, or encrypted junk)
```

FAT16:\title\00030017\484e41gg\data\private.sav:\menusave.dat (System Menu)
This private.sav file contains a 4000h-byte FAT12 image. The FAT12 contains only one file: menusave.dat (154h bytes), containing a list of Title IDs (and their sort-order how they are arranged in System Menu; users can drag the icons to rearrange their ordering):

```
0000h 4     ID ("TSSV")
0004h 4     Zerofilled (used somehow, can be nonzero?)
0008h 2     CRC16 on [000h..0153h], initial value 5356h, assume [008h]=0000h
000Ah 6     Zerofilled
0010h 39x8  Title IDs (gg,gg,gg,gg,tt,00,03,00) (0=NDS CartSlot or Unused)
0148h 8     Zerofilled
0150h 4     Index of NDS CartSlot Entry (0..39)
```

The current selection isn't stored in this file (instead, the Title ID of the most recently selected title is stored in the TWLCFGn.dat files).
Note that the "Nintendo Zone" utility isn't included in this list (even though it's present in title & ticket folders, and listed in wrap.bin).
The System Menu works even if data\private.sav doesn't exist (however, the sort-order is stored only if data\private.sav does exist).

DSi SD/MMC DSiware Files on External SD Card (.bin aka Tad Files)

The DSi can export applications from NAND to SD (via System Settings --> Data Managment). The ".bin" files created on SD are using the "Tad file structure", and alongsides, there's a "HNB_.lst" file containing a list of game codes.

SD:\private\ds\title\HNB_.lst (list of gamecodes)

```
000h 1200  List of 300 gamecodes, spelled backwards (or zero = unused entry)
4B0h 1     Language (0=Jap, 1=Eng, 2=Fre, 3=Ger, 4=Ita, 5=Spa, 6=Chi, 7=Kor?)
4B1h 3     Zero
4B4h 2     CRC16 on entries [000h..4B3h] (with initial value FFFFh)
4B6h 2     Zero
```

For example, "VUGK" in HNB_.lst would indicate gamecode KGUV aka "Flipnote Studio" for EUR/AUS regions. And the corresponding SD Card file would be "SD:\private\ds\title\4B475556.bin" (with the 4-letter gamecode encoded as 8-digit uppercase HEX number). The full 16-digit on eMMC storage would be "00030004\4b475556" (in lowercase, and with the "00030004" implied for the DSiWare folder; files from system folders cannot be exported to SD Card).
The Language byte reflects the System Settings's language selection at time when the HNB_.lst was created or modified (unknown why that info is stored in the file).

SD:\private\ds\title\4GGGGGGG.bin (encrypted executable/data in one file)

```
Offset   Size       Key  Description
000000h  4000h+20h  FIX  Icon/Title
004020h  B4h+20h    FIX  Header
0040F4h  440h+20h   FIX  Footer (certificates/hashes)
004554h  208h+20h   VAR  title.tmd (usually 208h bytes; but could be bigger)
00477Ch  size+N*20h VAR  000000vv.app
...      0          ?    seven N/A parts (unknown if/when they are used)
...      size+N*20h FIX  public.sav  (if any)
...      ?          ?    banner.sav  (if any)
```

ES Block Encryption is used to encrypt the header block, footer block, and the 11 content parts. Each are their own seperate ES blocks.

[DSi ES Block Encryption]

Some of the above blocks use fixed keys (FIX):

```
KEY_X[00h..0Fh] = Constant ("Nintendo DS",...)
KEY_Y[00h..0Fh] = Constant (66 82 32 04 ...)  ;from ARM7BIOS
since above X/Y are constant, that gives a fixed normal key:
KEY[00h..0Fh]   = Constant (3D A3 EA 33 ...)  ;as used in "dsi srl extract"
```

Other blocks use variable per-console keys (VAR):

```
KEY_X[00h..03h] = 4E00004Ah                          ;\
KEY_X[04h..07h] = 4A00004Eh                          ; same as for dev.kp
KEY_X[08h..0Bh] = Port[4004D00h+4] xor C80C4B72h     ;
KEY_X[0Ch..0Fh] = Port[4004D00h+0]                   ;/
KEY_Y[00h..0Fh] = Constant (CC FC A7 03 ...)  ;from ARM7BIOS
```

Without knowing the console-specifc Port[4004D00h] value, the data could be decrypted only by the DSi console that has originally exported the file to SD card.

However, Nintendo has somehow (maybe accidently) managed to store the Port[4004D00h] value as 16-digit ASCII string in the "TW cert"; which can be decrypted right from the SD card file (as done by the homebrew "dsi srl extract" utility).

Decrypted Icon/Title (at 0000h, size 4000h)

```
0000h 23C0h  Icon/Title (usually 23C0h bytes) ;see carthdr[068h,208h]
23C0h 1C40h  Zerofilled (padding to get 4000h byte size)
```

Decrypted Header block (at 4020h, size B4h)

```
000h  4      Fixed ID "4ANT" (aka TNA4, spelled backwards)
004h  2      Maker Code, spelled backwards ("10"=Nintendo) ;carthdr[010h]
006h  1      Zero
007h  1      Title version (vv)                            ;carthdr[01Eh]
008h  6      DSi MAC Address, spelled backwards            ;wifi_flash[036h]
00Eh  2      Zero
010h  16     Some console ID from HWINFO_N.dat             ;datfile[8Ch..9Bh]
020h  8      Title ID (gg gg gg gg 04 00 03 00)            ;carthdr[230h]
028h  4      Size of title.tmd    (usually 208h+20h)
02Ch  4      Size of 000000vv.app (size+N*20h)             ;carthdr[210h]
030h  4*7    Size of seven N/A parts (0)
04Ch  4      Size of public.sav   (size+N*20h)             ;carthdr[238h]
050h  4      Size of banner.sav?  (usually 0)       ;carthdr[1BFh].bit2=1
054h  8 * 4  List of eight Content IDs in same order as title.tmd
074h  0x3e   Reserved section per tmds, uh? (mostly zero, plus garbage?)
0B2h  2      Unknown (zero)
```

Decrypted Footer block (at 40F4h, size 460h)

```
000h  20     SHA1 of Icon/Title
014h  20     SHA1 of TNA4
028h  20     SHA1 of title.tmd
03Ch  20     SHA1 of 000000vv.app
040h  20*7   SHA1 of seven N/A parts (unused, can be whatever garbage)
0DCh  20     SHA1 of public.sav
0F0h  20     SHA1 of banner.sav
```

```
  104h  3Ch    ECC signature of [000h..103h] with AP cert
  140h  180h   AP cert, signed by TW cert
  2C0h  180h   TW cert, specific to a console (see dev.kp)
```
More in depth, the above two cert's look as so:
```
  140h 4    Signature Type (00,01,00,02) (ECC, sect233r1, non-RSA)  ;\
  144h 3Ch  Signature Hex numbers... across... below?               ; AP cert
  180h 40h  Signature padding/alignment (zerofilled)                ; 180h-byte
  1C0h 40h  Signature Name "Root-CA..-MS..-TW..-08..", 00h-padded   ;
            "Root-CA00000001-MS00000008-TWxxxxxxxx-08nnnnnnnnnnn1nn";
  200h 4    Key Type (00,00,00,02)       (ECC, sect233r1, non-RSA)  ;
  204h 40h  Key Name "AP00030015484e42gg", 00h-padded ;sys.settings ;
  244h 4    Key Random/time/type/flags/chksum?   ;<-- ZERO here     ;
  248h 3Ch  Key Public ECC Key (point X,Y) (random/per game?)       ;
  284h 3Ch  Key padding/alignment (zerofilled)                      ;/
  2C0h 4    Signature Type (00,01,00,02) (ECC, sect233r1, non-RSA)  ;\
  2C4h 3Ch  Signature Hex numbers... across... below?               ; TW cert
  300h 40h  Signature padding/alignment (zerofilled)                ; 180h-byte
  340h 40h  Signature Name "Root-CA00000001-MS00000008", 00h-padded ; (same as
  380h 4    Key Type (00,00,00,02)       (ECC, sect233r1, non-RSA)  ; dev.kp,
  384h 40h  Key Name "TWxxxxxxxx-08nnnnnnnnnnn1nn", 00h-padded      ; excluding
  3C4h 4    Key Random/time/type/flags/chksum?                      ; private
  3C8h 3Ch  Key Public ECC Key (point X,Y)                          ; key)
  404h 3Ch  Key padding/alignment (zerofilled)                      ;/
```

Much like the Wii, the DSi carries with it a private ECC key that it can use to sign things, and a certificate signed by Nintendo that attests to the fact that the public ECC key belongs to a genuine DSi.

DSi SD/MMC DSiware Files from Nintendo's Server

http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4ggggggg/000000vv
The "000000vv" file contains the ".app" file (in encrypted form).
```
  Server:  "000000vv"      (AES-CBC encrypted, raw)
  eMMC:    "000000vv.app"  (decrypted, raw)
  SD Card: "GGGGGGGG.bin"  (ES-block encrypted, with extra data)
```
First, the encrypted Title Key must be decrypted (via AES-CBC):
```
  KEY[00h..0Fh] = Common Key (AF,1B,F5,16,...)        ;from ARM7BIOS
  IV[00h..07h]  = Title ID (00,03,00,tt,gg,gg,gg,gg)  ;from tik/cetk[1DCh]
  IV[08h..0Fh]  = Zerofilled                          ;padding
  Input: Encrypted Title Key                          ;from tik/cetk[1BFh]
  Output: Decrypted Title Key                         ;for use in next step
```
Then, the actual executable/file can be decrypted (also via AES-CBC):
```
  KEY[00h..0Fh] = Decrypted Title Key                 ;from above step
  IV[00h..01h]  = Usually Zero (or "Index" from tmd?) ;from tmd[1E8h+N*24h] ?
  IV[02h..0Fh]  = Zerofilled                          ;padding
  Input: Encrypted file "000000vv"                    ;from http download
  Output: Decrypted file "000000vv.app"               ;saved on eMMC
```
The above decryption steps do require a big-endian AES-CBC software implementation (the DSi hardware supports only little-endian, and it supports only AES-CTR and AES-CCM, and, especially, it supports only the "encrypt" key schedule, whilst AES-CBC would require a different "decrypt" key schedule).

http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4ggggggg/cetk
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4ggggggg/tmd
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/000300tt4ggggggg/tmd.nn
These files do contain tickets ("cetk"), and title metadata ("tmd" for newest version, plus "tmd.nn" for older versions; with nn=0,1,256,257,512 and the like).
DSi SD/MMC DSiware Tickets and Title metadata
The "cetk" file is available only for free system updates (not for titles sold commercially in DSi Shop).

Downloading
The files can be downloaded with normal web browsers. The homebrew "NUS Downloader" utility is also allowing to download those files (and to decrypt them, provided that the "cetk" is available).
For free system updates, tickets can be downloaded as "cetk" files. For titles sold commercially in DSi ship, tickets must be purchased somehow differently.
For example, the updates for DSi System Settings (EUR) can be downloaded from:

```
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/00030015484e4250/tmd
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/00030015484e4250/cetk
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/00030015484e4250/00000002
http://nus.cdn.t.shop.nintendowifi.net/ccs/download/00030015484e4250/00000003
```

The tmd and cetk files are unencrypted. The 00000002 and 00000003 files are encrypted executables (containing System Settings v2 and v3).

Older Versions
Nintendo is usually keeping older "000000vv" files on their server, so one could still download those older versions.
The oldest/original version would be usually "00000000", however, in case of system files that are pre-installed on all consoles, only later versions are available as updates (ie. starting with "00000001", or for some reason, with "00000002", in case of system settings).
The tmd/cetk files are available only for the newest version (meaning that some cosmetic values like title size & title version are adjusted for the newest version; the cetk's AES-CBC key usually doesn't change for updates, so older versions can be still decrypted with newer cetk's).
Older versions are usually deleted on internal eMMC storage, so only the "000000vv.app" file exists.

DSi SD/MMC DSiware Tickets and Title metadata

Below describes the "raw" ticket+tmd formats. For more info on the data being stored/encrypted in various locations, see these chapters:
DSi SD/MMC DSiware Files on Internal eMMC Storage
DSi SD/MMC DSiware Files on External SD Card (.bin aka Tad Files)
DSi SD/MMC DSiware Files from Nintendo's Server

Ticket (cetk aka .tik)
Tickets exist as "cetk" files (as found on Nintendo's server), and as ".tik" files (as found in nand/ticket folder):

```
Server:  "cetk"         unencrypted, 2468 bytes (2A4h+700h), tik+certificate
eMMC:    "gggggggg.tik" encrypted,   708 bytes  (2A4h+20h),  tik+es_block
SD Card: N/A            N/A, tickets aren't exported to SD card
```

Tickets are used for decrypting downloads from DSi shop. They are essentially containing a 16-byte AES-CBC decryption key, plus signatures and some other stuff.

```
000h  4    Signature Type (00h,01h,00h,01h) (100h-byte RSA)
004h  100h Signature RSA-OpenPGP-SHA1 across 140h..2A3h
104h  3Ch  Signature padding/alignment (zerofilled)
140h  40h  Signature Name "Root-CA00000001-XS00000006", 00h-padded
180h  3Ch  ECDH data for one-time installation keys?   ;zero for free tik's
1BCh  3    Zero
1BFh  10h  Encrypted AES-CBC Title Key
1CFh  1    Zero
1D0h  8    Ticket ID  (00,03,xx,xx,xx,xx,xx,xx) ?
1D8h  4    Console ID (see dev.kp "TWxxxxxxxx", zero for free system updates)
1DCh  8    Title ID       (00,03,00,17,"HNAP")                  ;cart[230h]
1E4h  2    Zero (Wii: mostly FFFFh)
1E6h  2    Title Version (vv,00) (LITTLE-ENDIAN!?)       ;NEWEST ;cart[01Eh]
1E8h  4    Zero (Wii: Permitted Titles Mask)
1ECh  4    Zero (Wii: Permit mask)
1F0h  1    Zero (Wii: Allow Title Export using PRNG key, 0=No, 1=Yes)
1F1h  1    Zero (Wii: Common Key Index, 0=Normal, 1=Korea) (DSi: Always 0)
1F2h  2Fh  Zero
```

```
  221h  1    Unknown (01h) (Wii: Unknown, 00h=Non-VC, 01h=VC)
  222h  20h  FFh-filled (Wii: Content access permissions, 1 bit per content)
  242h  20h  00h-filled (Wii: Content access permissions, 1 bit per content)
  262h  2    Zero
  264h  4    Zero   ;Wii: Time Limit Enable  (0=Disable, 1=Enable)
  268h  4    Zero   ;Wii: Time Limit Seconds (uh, seconds since/till when?)
  26Ch  38h  Zero   ;Wii: Seven more Time Limits (Enable, Seconds)
  2A4h  700h Certificates (see below) (only in "cetk", not in ".tik)
```
The Launcher checks some of the permission entries, but it doesn't check RSA for tickets, so one can create own/dummy tickets. The Console ID and Title ID might be also unchecked, so one could possibly simply copy/rename existing tickets (from the same console), or decrypt/copy/re-encrypt tickets (from other consoles).
Note: It seems to be possible to store multiple tickets in one .tik file (in that case each ticket is separately encrypted in 2A4h+20h bytes).

Title metadata (tmd aka .tmd)
Title metadata exists as "tmd" file (as found on Nintendo's server), and as ".tmd" file (as found in eMMC title folders), and it's also included in ".bin" files (in files exported to SD cards):
```
  Server:  "tmd"          unencrypted, 2312 bytes (208h+700h), tmd+certificate
  Server:  "tmd.nn"       as above, OLDER tmd versions (nn=0,1,256,257,512,etc)
  eMMC:    "title.tmd"    unencrypted, 520 bytes  (208h+0),    tmd
  SD Card: "GGGGGGGG.bin" encrypted, huge file, contains .app+tmd+sav files
```
Title metadata contains signatures and other useless stuff. One possibly useful feature is that it allows to define more than one "content" per title, however, that feature appears to be only used on Wii. DSi titles are usually having only one content (the ".app" file).
```
  000h        4    Signature Type (00h,01h,00h,01h) (100h-byte RSA)
  004h        100h Signature RSA-OpenPGP-SHA1 across 140h..207h
  104h        3Ch  Signature padding/alignment (zerofilled)
  140h        40h  Signature Name "Root-CA00000001-CP00000007", 00h-padded
  180h        1    Version               (00h)
  181h        1    ca_crl_version        (00h)
  182h        1    signer_crl_version    (00h)
  183h        1    Zero       (padding/align 4h)
  184h        8    System Version (0)
  18Ch        8    Title ID       (00,03,00,17,"HNAP")            ;cart[230h]
  194h        4    Title Type     (0)
  198h        2    Group ID       (eg. "01"=Nintendo)             ;cart[010h]
  19Ah        4    SD/MMC "public.sav" filesize in bytes  (0=none) ;cart[238h]
  19Eh        4    SD/MMC "private.sav" filesize in bytes (0=none) ;cart[23Ch]
  1A2h        8    Zerofilled
  1AAh        10h  Parental Control Age Ratings                    ;cart[2F0h]
  1BAh        1Eh  Zerofilled
  1D8h        4    Access rights (0)
  1DCh        2    Title Version (vv,00) (LITTLE-ENDIAN!?) ;NEWEST ;cart[01Eh]
  1DEh        2    Number of contents (at 1E4h and up) (usually 00h,01h)
  1E0h        2    boot index    (0)
  1E2h        2    Zerofilled (padding/align 4h)
  1E4h+N*24h  4    Content ID    (00,00,00,vv) ;lowercase/hex ;"0000000vv.app"
  1E8h+N*24h  2    Content Index (00,00)
  1EAh+N*24h  2    Content Type  (00,01)  ;aka DSi .app
  1ECh+N*24h  8    Content Size  (00,00,00,00,00,19,E4,00) ;NEWEST ;cart[210h]
  1F4h+N*24h  14h  Content SHA1  (on decrypted ".app" file);NEWEST
  208h+..     700h Certificates (see below) (only in "tmd", not in ".tmd)
```
The Launcher does verify the .tmd's RSA signature, and uses the Title/Content IDs to create the path/filename for the .app file. The Version, Size, SHA1 entries are not verified, so one could use any .tmd version with any .app version (when renaming the .app to match the .tmd's Content ID, but that'd be a messy solution, and it's better to use the correct .tmd per .app).
Note: title.tmd is usually/always 208h bytes (one content), max permitted size is 49E4h (200h contents), a larger filesize can crash the firmware (used by Unlaunch.dsi exploit).

Certificates (at end of "cetk" and "tmd) (not in ".tik" or ".tmd)
```
  cert cetk tmd  siz  content
```

```
  000h 2A4h 208h 4    Signature Type (00h,01h,00h,01h)            ;\
  004h 2A8h 20Ch 100h Signature                                   ;
  104h 3A8h 30Ch 3Ch  Signature padding/alignment (zerofilled)    ;
  140h 3E4h 348h 40h  Signature Name "Root-CA00000001", 00h-padded ; 300h bytes
  180h 424h 388h 4    Key Type (00,00,00,01) (100h-byte RSA)      ;
  184h 428h 38Ch 40h  Key Name "XS00000006", 00h-padded           ;
  1C4h 468h 3CCh 4    Key Random/time/type/flags/chksum?          ;
  1C8h 46Ch 3D0h 100h Key Public RSA Key                          ;
  2C8h 56Ch 4D0h 4    Key Public RSA Exponent? (00,01,00,01)      ;
  2CCh 570h 4D4h 34h  Key padding/alignment (zerofilled)          ;/
  300h 5A4h 508h 4    Signature Type (00h,01h,00h,00h)            ;\
  304h 5A8h 50Ch 200h Signature                                   ;
  504h 7A8h 70Ch 3Ch  Signature padding/alignment (zerofilled)    ;
  540h 7E4h 748h 40h  Signature Name "Root" (padded with 00h)     ; 400h bytes
  580h 824h 788h 4    Key Type (00,00,00,01) (100h-byte RSA)      ;
  584h 828h 78Ch 40h  Key Name "CA00000001", 00h-padded           ;
  5C4h 868h 7CCh 4    Key Random/time/type/flags/chksum?          ;
  5C8h 86Ch 7D0h 100h Key Public RSA Key                          ;
  6C8h 86Ch 8D0h 4    Key Public RSA Exponent? (00,01,00,01)      ;
  6CCh 970h 8D4h 34h  Key padding/alignment (zerofilled)          ;/
```

Note: Above certificates should be just a 1:1 copy of the entries in cert.sys.
Note: "XS00000006" or "XS00000003" is found in cetk/tik's, in tmd's it's called "CP00000007".

Notes
The Title Version and Content Size/SHA1 entries are reflecting the NEWEST ".app" version (but the AES-CBC key should usually/always also work for older versions).
The homebrew NUS Downloader utility is saving TMD (520 bytes; with REMOVED certificate) and CETK (2468 bytes; with included certificate).

DSi SD/MMC Firmware dev.kp and cert.sys Certificate Files

FAT16:\sys\cert.sys ;3904 bytes (or 2560 bytes before DSi-Shop connection)
Data in this file is same on all retail DSi consoles (even for different regions like US and EUR and KOR).

```
  000h 300h  Public RSA Key "XS00000006" signed by "Root-CA00000001"
  300h 400h  Public RSA Key "CA00000001" signed by "Root"
  700h 300h  Public RSA Key "CP00000007" signed by "Root-CA00000001"
 Below NOT in Korea? Or NOT when notyet connected to DSi Shop?
  A00h 240h  Public ECC Key "MS00000008" signed by "Root-CA00000001"
  C40h 300h  Public RSA Key "XS00000003" signed by "Root-CA00000001"
```

The cert.sys for DSi debug version is different:

```
  000h 300h  Public RSA Key "CP00000005" signed by "Root-CA00000002"
  300h 300h  Public RSA Key "XS00000006" signed by "Root-CA00000002"
  600h 400h  Public RSA Key "CA00000002" signed by "Root"
  A00h 300h  Public RSA Key "CP00000007" signed by "Root-CA00000002"
```

More detailed, the retail version of cert.sys looks as so:

```
  000h 4    Signature Type (00,01,00,01) (100h-byte RSA)        ;\
  004h 100h Signature RSA-OpenPGP-SHA1 across 140h..2FF         ;
  104h 3Ch  Signature padding/alignment (zerofilled)            ;
  140h 40h  Signature Name "Root-CA00000001", 00h-padded        ;
  180h 4    Key Type (00,00,00,01) (100h-byte RSA)              ;
  184h 40h  Key Name "XS00000006", 00h-padded                   ;
  1C4h 4    Key Random/time/type/flags/chksum?                  ;
  1C8h 100h Key Public RSA Key (92,FF,96,40..)                  ;
  2C8h 4    Key Public RSA Exponent? (00,01,00,01)              ;
  2CCh 34h  Key padding/alignment (zerofilled)                  ;/
  300h 4    Signature Type (00,01,00,00) (200h-byte RSA) (!)    ;\
  304h 200h Signature RSA-OpenPGP-SHA1 across 540h..6FF         ;
  504h 3Ch  Signature padding/alignment (zerofilled)            ;
  540h 40h  Signature Name "Root", 00h-padded                   ;
  580h 4    Key Type (00,00,00,01) (100h-byte RSA)              ;
```

```
  584h 40h  Key Name "CA00000001", 00h-padded                   ;
  5C4h 4    Key Random/time/type/flags/chksum?                  ;
  5C8h 100h Key Public RSA Key (B2,79,C9,E2..)                  ;
  6C8h 4    Key Public RSA Exponent? (00,01,00,01)              ;
  6CCh 34h  Key padding/alignment (zerofilled)                  ;/
  700h 4    Signature Type (00,01,00,00) (100h-byte RSA)        ;\
  704h 100h Signature RSA-OpenPGP-SHA1 across 840h..9FF         ;
  804h 3Ch  Signature padding/alignment (zerofilled)            ;
  840h 40h  Signature Name "Root-CA00000001", 00h-padded        ;
  880h 4    Key Type (00,00,00,01) (100h-byte RSA)              ;
  884h 40h  Key Name "CP00000007", 00h-padded                   ;
  8C4h 4    Key Random/time/type/flags/chksum?                  ;
  8C8h 100h Key Public RSA Key (93,BC,0D,1F..)                  ;
  9C8h 4    Key Public RSA Exponent? (00,01,00,01)              ;
  9CCh 34h  Key padding/alignment (zerofilled)                  ;/
 Below NOT when notyet connected to DSi Shop:
  A00h 4    Signature Type (00,01,00,01) (100h-byte RSA)        ;\
  A04h 100h Signature RSA-OpenPGP-SHA1 across B40h..C3F         ;
  B04h 3Ch  Signature padding/alignment (zerofilled)            ;
  B40h 40h  Signature Name "Root-CA00000001", 00h-padded        ;
  B80h 4    Key Type (00,00,00,02)   (ECC, sect233r1, non-RSA)  ;
  B84h 40h  Key Name "MS00000008", 00h-padded                   ;
  BC4h 4    Key Random/time/type/flags/chksum?                  ;
  BC8h 3Ch  Key Public ECC Key (point X,Y) (01,93,6D,08..)      ;
  C04h 3Ch  Key padding/alignment (zerofilled)                  ;/
  C40h 4    Signature Type (00,01,00,01) (100h-byte RSA)        ;\
  C44h 100h Signature RSA-OpenPGP-SHA1 across D80h..F3F         ;
  D44h 3Ch  Signature padding/alignment (zerofilled)            ;
  D80h 40h  Signature Name "Root-CA00000001", 00h-padded        ;
  DC0h 4    Key Type (00,00,00,01) (100h-byte RSA)              ;
  DC4h 40h  Key Name "XS00000003", 00h-padded                   ;
  D04h 4    Key Random/time/type/flags/chksum?                  ;
  E08h 100h Key Public RSA Key (AD,07,A9,37..)                  ;
  F08h 4    Key Public RSA Exponent? (00,01,00,01)              ;
  F0Ch 34h  Key padding/alignment (zerofilled)                  ;/
```

Cert chain for the DSi. Contains certificates with signed keys:

```
  Root-CA00000001:            used for signing the four certificates below
  Root-CA00000001-CP00000007: used for signing TMDs ("Content Protection"?)
  Root-CA00000001-MS00000008: used for signing per-console ECC keys ("Master"?)
  Root-CA00000001-XS00000003: used for signing tickets from the DSiWare Shop
  Root-CA00000001-XS00000006: used for signing (common) tickets ("access"?)
```

A similar file with the same name exists on the Wii.

FAT16:\sys\dev.kp ;446 bytes (encrypted), 414 bytes (when decrypted)

The dev.kp file is encrypted with ES Block Encryption (using same key X/Y as for .tik files):

```
  KEY_X[00h..03h] = 4E00004Ah                           ;\
  KEY_X[04h..07h] = 4A00004Eh                           ; same as for Tad
  KEY_X[08h..0Bh] = Port[4004D00h+4] xor C80C4B72h      ;
  KEY_X[0Ch..0Fh] = Port[4004D00h+0]                    ;/
  KEY_Y[00h..0Fh] = Constant (E5,CC,5A,8B,...)  ;from ARM7BIOS
```

[DSi ES Block Encryption]

The decrypted dev.kp contains following entries:

```
  000h 4    Signature Type (00,01,00,02) (ECC, sect233r1, non-RSA)  ;\
  004h 3Ch  Signature Hex numbers... across... below?               ;
  040h 40h  Signature padding/alignment (zerofilled)                ;
  080h 40h  Signature Name "Root-CA00000001-MS00000008", 00h-padded ;
  0C0h 4    Key Type (00,00,00,02)       (ECC, sect233r1, non-RSA)  ;
  0C4h 40h  Key Name "TWxxxxxxxx-08nnnnnnnnnn1nn", 00h-padded       ;
  104h 4    Key Random/time/type/flags/chksum?                      ;
  108h 3Ch  Key Public ECC Key (point X,Y)       ;<-- public key    ;
  144h 3Ch  Key padding/alignment (zerofilled)                      ;
  180h 1Eh  Key Private ECC Key                  ;<-- private key   ;/
```

The "TWxxxxxxxx-08nnnnnnnnnn1nn" string may vary:

```
  "TW" might be for DSi only (ie. it might be different on DSi XL or 3DS?)
  "xxxxxxxx" is 8-digit lower-case hex number (unknown where from; for .tik)
  "08nnnnnnnnnnn1nn" is 16-digit lower-case hex number (from Port 4004D00h)
```

Example:

```
  Signature across rest of block -- type = 0x00010002, ECC
  0000000: 00 01 00 02 00 db da 21 3b e1 f1 bf bb 4d dc 1d
  0000010: 60 29 da 19 42 1e 66 4f a8 e5 27 a1 d4 ea 46 7d
  0000020: 9b b4 00 95 c5 0d e8 fa ef a7 8d e9 bc 54 da c1
  0000030: 24 94 0b 7c ad a8 61 d5 05 97 c2 64 38 ad 18 f9

  0000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

  Key used to sign this cert (Root-CA00000001-MS00000008)
  0000080: 52 6f 6f 74 2d 43 41 30 30 30 30 30 30 31 2d  Root-CA00000001-
  0000090: 4d 53 30 30 30 30 30 30 30 38 00 00 00 00 00 00  MS00000008
  00000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

  Console ID string
  00000c0: 00 00 00 02 54 57 63 37 39 64 63 65 63 39 2d 30  ....TWc79dcec9-0
  00000d0: 38 61 32 30 32 38 37 30 31 30 38 34 31 31 38 00  8a2028701084118.
  00000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

  Public ECC key (30 bytes, starting at 0x108)
  0000100: 00 00 00 00 6f dd de 42 01 e0 34 a3 19 bc a9 af
  0000110: 50 fe 8a ac 75 08 07 a9 3a 2c 21 51 93 ae 4a 90
  0000120: 6e 62 41 f1 a2 fe 00 00 3d 0a 13 97 da 53 17 98
  0000130: 69 38 65 67 ca f4 9c 87 ec 44 b7 eb d0 ec b8 3d
  0000140: 23 cf 7a 35 00 00 00 00 00 00 00 00 00 00 00 00
  0000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

  Private per-console ECC key, used for signing files on SD
  0000180: 01 12 9d e0 77 82 44 d3 ee 99 ad ce e5 fa fa ed
  0000190: c9 ab 8e a1 f9 b5 c8 14 3c 74 74 f8 19 3a
```

Note that the console id itself is burned in an OTP area of the TWL CPU, and changing the contents of this file will not actually change the console id.

This file contains the unique per-console ECC private-public key pair, along with a certificate issued by Nintendo.

This file is created by the DSi Shop, with data from a SOAP reply. The SOAP request data includes the hw console id, and the 0x100-byte RSA signature stored in NAND file "HWID.sgn". Trying to send that request would require a NAND dump, but when you have a NAND dump already sending that request is pointless since you can grab dev.kp from NAND.

Sending that request is pointless anyway since the dev.kp data from the server is random. The returned dev.kp data from the server for the EC private/public keys are random, the ticket consoleID immediately following TW before - in the twcert keyid is random as well. DSi Shop and System Settings don't contain any code for deleting dev.kp. If you try to delete/rename dev.kp manually from NAND a new dev.kp will be generated by the shop, but then the server will return an error since the server account public dev.kp cert won't match.

Data management can't be accessed when dev.kp doesn't exist since you'd have no twcert to sign/verify tads with, like when you never connected the DSi Shop server.

DSi SD/MMC Firmware Font File

FAT16:\sys\TWLFontTable.dat ;843.1K (D2C40h bytes) (compressed) (Normal)
FAT16:\sys\TWLFontTable.dat ;568.1K (8E020h bytes) (compressed) (China)
FAT16:\sys\TWLFontTable.dat ;158.9K (27B80h bytes) (compressed) (Korea)
This file contains LZrev-compressed fonts in the NFTR (Nitro font) format.
This is the only real long filename that exceeds the 8.3 limit on the DSi (alternate short name is TWLFON~1.DAT). DSi software is often using a virtual filename "nand:/<sharedFont>" on ARM9 side, which is then replaced by "nand:/sys/TWLFontTable.dat" on ARM7 side.
The font can be used by DSiware titles (DSi ROM cartridges usually don't have eMMC access, and thus cannot use the font).

```
  0000h       80h  RSA-SHA1 on entries [0080h..009Fh] (23h,8Bh,F9h,08h,...)
  0080h       4    Date? (00h,31h,07h,08h=Normal, 00h,27h,05h,09h=China/Korea)
  0084h       1    Number of NFTR resources (3=Normal, 9=China/Korea)
  0085h       1    Zerofilled
  0086h       1    Unknown (0=Normal, 4=China, 5=Korea)
  0087h       5    Zerofilled
  008Ch       14h  SHA1 on below resource headers at [00A0h+(0..NUM*40h-1)]
  00A0h+N*40h 20h  Resource Name in ASCII, padded with 00h
  00C0h+N*40h 4    Compressed Resource Size in .dat file   ;\compressed
  00C4h+N*40h 4    Compressed Resource Start in .dat file  ;/
  00C8h+N*40h 4    Decompressed Resource Size              ;-decompressed
  00CCh+N*40h 14h  SHA1 on Compressed Resource at [Start+0..Size-1]
  ...         ..   Compressed Font Resources (with 16-byte alignment padding)
```

The resources are containing the same font thrice (at three different sizes), normal consoles have only three resources (0,1,2), chinese/korean consoles have nine resources (six zerofilled 40h-byte entries, and three actually used entries: 3,4,5 for china, or 6,7,8 for korea). The name strings of the resources are:

```
  "TBF1_l.NFTR"     ;0 Large  16x21 pixels ;\Normal (blurry: 4 colors used)
  "TBF1_m.NFTR"     ;1 Medium 12x16 pixels ; 2bpp, 7365 characters, Unicode
  "TBF1_s.NFTR"     ;2 Small  10x12 pixels ;/
  "TBF1-cn_l.NFTR"  ;3 Large  16x21 pixels ;\China (crisp-clear: 2 colors used)
  "TBF1-cn_m.NFTR"  ;4 Medium 12x16 pixels ; 2bpp, 7848 characters, Unicode
  "TBF1-cn_s.NFTR"  ;5 Small  12x13 pixels ;/
  "TBF1-kr_l.NFTR"  ;6 Large  16x21 pixels ;\Korea (crisp-clear: 2 colors used)
  "TBF1-kr_m.NFTR"  ;7 Medium 12x16 pixels ; 2bpp, 3679 characters, Unicode
  "TBF1-kr_s.NFTR"  ;8 Small  12x12 pixels ;/
```

All characters have proportional width (as defined in the Character Width chunk), eg. the width of the 16x21 font can be max 16 pixels (plus spacing), but most of the characters are less than 16 pixels wide.
The character numbers in the Char Map chunks are 16bit Unicode, supporting ASCII, plus extra punctuation marks, european letters with accent marks, greek, cyrillic, math symbols, and thousands of japanese letters. There are also some custom nintendo-specific symbols (like buttons and Wii symbols).

Notes
The blurry japanese letters aren't actually legible (especially for the smaller font sizes). Also note that the size of small font varies for the three regions (ie. 10x12, 12x13, or 12x12).

LZrev Compressed Font Resource Format

```
  ..   uncompressed area (usually 15h bytes)
  ...  compressed area (decompressed backwards)
  ..   footer: padding (to 4-byte boundary)
  3    footer: size of footer+compressed area (offset to compressed.bottom)
  1    footer: size of footer (offset to compressed.top)
  4    footer: extra DEST size (offset to decompressed.top)
  ..   zeropadding to 10h-byte boundary
```

LZ Decompression Functions
The original decompression function can be found in Flipnote (EUR) at address 20BF8E4h (which is mainly doing error checking, and then calling the actual decompression function at 20BF938h) (Flipnote does also contain several custom fonts, the TWLFontTable.dat file is used only for Flipnote's "Help" function).

Nitro Font Resources

The format of the decompressed data is Nintendo's standard Nitro Font format:
DS Cartridge Nitro Font Resource Format

DS Cartridge Nitro Font Resource Format

Nitro Font Resource File formats (compressed & uncompressed)
Nitro Fonts are often found as .NFTR or .ZFTR files (within NitroROM filesystems). The DSi firmware does additionally contain Nitro Fonts in a .dat file (in the eMMC FAT16 filesystem).

```
.NFTR  Raw uncompressed Nitro Font Resource
.ZFTR  LZ11-compressed Nitro Font Resource
.dat   Archive with three LZrev-compressed Nitro Font Resources (used on DSi)
```

The .ZFTR files are containing a 4-byte compression header, followed by the compressed data (starting with the first compression flag byte, following by the first chunk header). For details, see:
LZ Decompression Functions
The DSi's "\sys\TWLFontTable.dat" contains three fonts, using a special LZ compression variant (with the data decompressed backwards, starting at highest memory address). For details, see:
DSi SD/MMC Firmware Font File
Either way, the decompressed fonts are looking as follows:

Nitro Font Resource Header Chunk

```
00h 4    Chunk ID "RTFN" (Nitro Font Resource)
04h 2    Byte Order    (FEFFh) (indicates that above is to be read backwards)
06h 2    Version       (0100h..0102h) (usually 0101h or 0102h)
08h 4    Decompressed Resource Size (000A3278h) (including the NFTR header)
0Ch 2    Offset to "FNIF" Chunk, aka Size of "RTFN" Chunk (0010h)
0Eh 2    Total number of following Chunks (0003h+NumCharMaps) (0018h)
```

Font Info Chunk

```
00h      4    Chunk ID "FNIF" (Font Info)
04h      4    Chunk Size (1Ch or 20h)
08h      1    Unknown/unused (zero)
09h xxx  1    Height                        ;or Height+/-1
0Ah xxx  1    Unknown (usually 00h, or sometimes 1Fh)
0Bh      2    Unknown/unused (zero)
0Dh xxx  1    Width           ;\or Width+1
0Eh xxx  1    Width_bis (?)   ;/
0Fh      1    Encoding (0=UTF8, 1=Unicode, 2=SJIS, 3=CP1252) (usually 1)
10h      4    Offset to Character Glyph chunk, plus 8
14h      4    Offset to Character Width chunk, plus 8
18h      4    Offset to first Character Map chunk, plus 8
1Ch      (1)  Tile Height                                ;\present only
1Dh xxx  (1)  Max Width or so +/-?                       ; when above
1Eh      (1)  Underline location                         ; Chunk Size = 20h
1Fh      (1)  Unknown/unused (zero)                      ;/(version 0102h)
```

Character Glyph (Tile Bitmaps)

```
00h      4    Chunk ID "PLGC" (Character Glyph)
04h      4    Chunk Size (10h+NumTiles*siz+padding)
08h      1    Tile Width in pixels
09h      1    Tile Height in pixels
0Ah      2    Tile Size in bytes (siz=width*height*bpp+7)/8)
0Ch      1    Underline location
0Dh      1    Max proportional Width including left/right spacing
0Eh      1    Tile Depth (bits per pixel) (usually 1 or 2, sometimes 3)
0Fh      1    Tile Rotation (0=None/normal, other=see below)
10h      ...  Tile Bitmaps
...      ...  Padding to 4-byte boundary (zerofilled)
```

All tiles are starting on a byte boundary. However, the separate scanlines aren't necessarily byte-aligned (for example, at 10pix width, a byte may contain rightmost pixels of one line, followed by leftmost pixels of next line).
Bit7 of the first byte of a bitmap is the MSB of the upper-left pixel, bit6..0 are then containing the LSB(s) of the pixel (if bpp>1), followed by the next pixels of the scanline, followed by further scanlines; the data is arranged as straight Width*Height bitmap (without splitting into 8x8 sub-tiles).
Colors are ranging from Zero (transparent/background color) to all bit(s) set (solid/text color).
The meaning of the Tile Rotation entry is unclear (one source claims 0=0', 1=90', 2=270', 3=180', and another source claims 0=0', 2=90', 4=180', 6=270', and for both sources, it's unclear if the rotation is meant to be clockwise or anti-clockwise).

Character Width

```
00h      4    Chunk ID "HDWC" (Character Width)
04h      4    Chunk Size (10h+NumTiles*3+padding)
08h      2    First Tile Number (should be 0000h)
0Ah      2    Last Tile Number  (should be NumTiles-1)
0Ch      4    Unknown/unused (zero)
10h+N*3  1    Left Spacing (to be inserted left of character bitmap)
11h+N*3  1    Width of Character Bitmap (excluding left/right spacing)
12h+N*3  1    Total Width of Character (including left/right spacing)
...      ...  Padding to 4-byte boundary (zerofilled)
```

Defines the proportional character width for each tile. Entry [11h+N*3] defines the width of the non-transparent character area (which left-aligned in the Tile Bitmap; any further pixels in the Bitmap are unused/zero). The other two entries define the left/right spacing that is needed to be added to the character.

Character Map(s) - Translation Tables for ASCII/JIS/etc to Tile Numbers?

```
00h      4    Chunk ID "PAMC" (Character Map)
04h      4    Chunk Size (14h+...+padding)
08h      2    First Character (eg. 0020h=First ASCII Char)
0Ah      2    Last Character  (eg. 007Eh=Last ASCII Char)
0Ch      4    Map Type (0..2, for entry 14h and up, see there)
10h      4    Offset to next Character Map, plus 8   (0=None, no further)
```

For Map Type0, Increasing TileNo's assigned to increasing CharNo's:

```
14h      2    TileNo for First Char (and increasing for further chars)
16h      2    Padding to 4-byte boundary (zerofilled)
```

For Map Type1, Custom TileNo's assigned to increasing CharNo's:

```
14h+N*2  2    TileNo's for First..Last Char (FFFFh=None; no tile assigned)
...      0/2  Padding to 4-byte boundary (zerofilled)
```

For Map Type2, Custom TileNo's assigned to custom CharNo's:

```
14h      2    Number of following Char=Tile groups...
16h+N*4  2    Character Number
18h+N*4  2    Tile Number
...      2    Padding to 4-byte boundary (zerofilled)
```

These chunks are containing tables for translating character numbers (eg. Unicode numbers, or whatever format is selected in the Font Info chunk) to actual Tile Numbers (ie. the way how tiles are ordered in the Glyph and Width chunks).
Font files can contain several Character Map chunks (eg. some Type0 chunks for Char 0020h..007Eh and Char 00A0h..00FFh, plus some Type1 chunks for areas like Char 037Eh..0451h, plus one large Type2 chunk for everything that wasn't defined in the other chunks; the First/Last Character entries are don't care for Type2, they are usually set to First=0000h and Last=FFFFh in that case). Characters that are NOT included in any of the tables should be treated as undefined (as so for any characters that are assigned as Tile=FFFFh in Type1 chunks).
Unicode character numbers are stored as 16bit values. Unknown how other character numbers like UTF8 or SJIS are stored.

LZ Decompression Functions

LZSS and LZ11 - Decompression for BIOS/SWI and ZFTR Font Files

Below function can decompress LZSS data (as used by BIOS SWIs), and the LZ11 variant (with 11h in the header; as used by ZFTR font files, but not being compatible with the BIOS SWI functions on neither GBA nor NDS nor DSi).

```
  typ=byte[src], fin=dst+(word[src]/100h), src=src+4
 @@collect_more:
  flagbits=[src], src=src+1, numflags=8
 @@decompress_lop:
  if dst>=fin then goto @@decompress_done
  if numflags=0 then goto @@collect_more
  numflags=numflags-1, flagbits=flagbits*2
  if (flagbits AND 100h)=0 then
    [dst]=[src], dst=dst+1, src=src+1
  else
    if typ=10h        ;LZ10 aka LZSS (BIOS SWI compatible)
      len=3
    elseif typ=11h    ;LZ11 (special extended format)
      if [src]/10h>1 then len=001h
      if [src]/10h<1 then len=011h+([src] AND 0Fh)*10h, src=src+1
      if [src]/10h=1 then len=111h+([src] AND 0Fh)*1000h+[src+1]*10h, src=src+2
    endif
    len=len+[src]/10h, disp=001h+([src] AND 0Fh)*100h+[src+1], src=src+2
    for i=1 to len, [dst]=[dst-disp], dst=dst+1, next i
  endif
  goto @@decompress_lop
 @@decompress_done:
  ret
```

The LZSS variant (but not LZ11) can be decompressed by BIOS SWI functions:
BIOS Decompression Functions

LZrev - Reverse Decompression

```
  Used for DSi Font (TWLFontTable.dat)
  Used for 3DS .code files (within NCCH ExeFS filesystems)
```

This function resembles LZSS, but with src/dst processed in reversed order (starting at highest memory location). Further differences are that the header is replaced by a footer, and, weirdly, data is copied from "disp+3" instead of "disp+1", and, the decompression is intended to use a single buffer for src/dst (so the source data will be overwritten during decompression) (in order to avoid overwriting still unprocessed data, some bytes near start of file are usually left uncompressed, eg. for the font files, decompression usually ends somewhere at fin=buf+15h rather than at fin=buf+0).

```
  dest_size=src_size+[src+src_size-4]   ;when dest_size is unknown (for 3DS)
  allocate buf(dest_size), copy "src_size" bytes from file to buf
  src=buf+src_size                      ;origin = pointing after footer
  dst=src+(word[src-4])-1               ;dst = src plus extra len
  fin=src-(word[src-8] AND 00FFFFFFh)   ;fin = src minus compressed_len
  src=src-(byte[src-5])-1               ;src = src minus footer_len
 @@collect_more:
  flagbits=[src], src=src-1, numflags=8
 @@decompress_lop:
  if src<=fin then goto @@decompress_done
  if numflags=0 then goto @@collect_more
  numflags=numflags-1, flagbits=flagbits*2
  if (flagbits AND 100h)=0 then
    [dst]=[src], dst=dst-1, src=src-1
  else
    len=([src]/10h)+3, disp=([src] AND 0Fh)*100h+([src-1])+3, src=src-2
    for i=1 to len, [dst]=[dst+disp], dst=dst-1, next i
  endif
  goto @@decompress_lop
 @@decompress_done:
  ret
```

Note: The footer_len at [src-5] is usually 08h+0..3 (with 0..3 padding bytes to make the footer entries & file size word aligned).

DSi SD/MMC Firmware Log Files

FAT16:\sys\log\shop.log (32 bytes)

```
0000h 20h   Zerofilled
```

Unknown if this file can contain anything else.

FAT16:\sys\log\product.log (573 bytes)

Contains some ASCII text with version, date (YY/MM/DD), and time (HH:SS) info (using 0Ah as CRLF):

```
0,BOARD,START,1.5,09/01/14,14:52,000055, ,
0,BOARD,OK,1.5,09/01/14,14:53,000055, ,
0,TP_CAL,OK,2.0, , , ,(647 811)-(3478 3245),
0,AGING,OK,1.0, , , ,Time=60:20(m:s)
Count=32,
0,FINAL,START,1.5,09/01/15,09:52,000084,TWL Ver.2.0,
0,FINAL,OK,1.5,09/01/15,09:52,000084,TWL Ver.2.0,
0,MIC,OK,2.1, , , ,All Test Passed,
0,CAMERA,OK,2.1, , , , ,
0,WRFU,START,0.60,09/01/15,10:03,000143,P000063 G000143 717cfde74f5ef6763473,
0,WRFU,OK,0.60,09/01/15,10:04,000143,PCVer:1.7f
R-53 -55 E0.00 0.00,
0,IMPORT,START,1.0, , , , ,
0,IMPORT,OK,1.0, , , ,Region=EUR,
0,NCHECK,OK,1.0, , , , ,
```

FAT16:\sys\log\sysmenu.log (16Kbytes)

Contains several groups of three text lines. Each "#FFT" group begins with two 0Ah characters, and is followed by space padding for 256-byte alignment of the next group).

Below is some example (with blank space removed, original 40-digit hash strings abbreviated to "xxxx..xx", and some lines are replaced by "..."):

```
#FFT 13-08-18[SUN] 12:37:10
title: HNAP
DHT_PAHSE1_FAILED (sub info): hash1     - 8dfc..59
#FFT 13-08-18[SUN] 12:37:10
title: HNAP
DHT_PAHSE1_FAILED (sub info): calc_hash - 7eca..f5
#FFT 13-08-18[SUN] 12:37:11
title: HNAP
menuRedIplManager.cpp [l.514] RED FATAL 0000000010000000 (0000000041575445)
#FFT 13-08-18[SUN] 12:37:11
title: HNAP
menuResetCallback.cpp [l.50] type 0
#FFT 13-08-18[SUN] 13:44:16
title: HNAP
DHT_PAHSE1_FAILED (sub info): hash1Addr-02799e38
#FFT 13-08-18[SUN] 13:44:16
title: HNAP
DHT_PAHSE1_FAILED (sub info): hash1     - 8dfc..59
...
...
#FFT 13-09-10[TUE] 22:07:39
title: HNAP
menuResetCallback.cpp [l.50] type 0
#FFT 13-09-14[SAT] 14:59:16
title: HNAP
SYSMi_LoadTitleThreadFunc: some error has occurred.
#FFT 13-09-14[SAT] 14:59:16
title: HNAP
```

```
SYSMi_AuthenticateTitleThreadFunc: loaded 1 times.
#FFT 13-09-14[SAT] 14:59:17
title: HNAP
menuRedIplManager.cpp [l.514] RED FATAL 0000800000002100 (0003000049524544)
#FFT 13-09-14[SAT] 14:59:17
title: HNAP
menuResetCallback.cpp [l.50] type 0
#FFT 00-01-03[MON] 20:50:18
title: HNAP
WHITELIST_NOTFOUND (sub info): no entry for phase 1/2.
#FFT 00-01-03[MON] 20:50:18
title: HNAP
WHITELIST_NOTFOUND (sub info): no entry for phase 3.
#FFT 00-01-03[MON] 20:50:18
title: HNAP
SYSMi_LoadTitleThreadFunc: some error has occurred.
#FFT 00-01-03[MON] 20:50:18
title: HNAP
SYSMi_AuthenticateTitleThreadFunc: loaded 1 times.
#FFT 00-01-03[MON] 20:50:19
title: HNAP
menuRedIplManager.cpp [l.514] RED FATAL 0000800008000100 (000000004143454b)
#FFT 00-01-03[MON] 20:50:19
title: HNAP
menuResetCallback.cpp [l.50] type 0
#FFT 00-01-05[WED] 01:03:16
title: HNAP
WHITELIST_NOTFOUND (sub info): no entry for phase 1/2.
...
...
#FFT 00-01-01[SAT] 00:02:37
title: HNAP
SYSMi_AuthenticateTitleThreadFunc: loaded 1 times.
#FFT 00-01-01[SAT] 00:02:38
title: HNAP
menuRedIplManager.cpp [l.514] RED FATAL 0002004000000100 (00000000414e5045)
#FFT 00-01-01[SAT] 00:02:38
title: HNAP
menuResetCallback.cpp [l.50] type 0
```

DSi SD/MMC Firmware Misc Files

FAT16:\sys\HWINFO_S.dat (aka Serial/Barcode) (16Kbytes)

```
0000h 80h   RSA-SHA1-HMAC across entries [0088h..00A3h]
            (with RSA key from Bootsectors, and also from Launcher)
            (with SHA1-HMAC key = SHA1([4004D00h..4004D07h], aka Console ID)
0080h 4     Header, Version or so (00000001h)
0084h 4     Header, Size of entries at [0088h..00A3h] (0000001Ch)
0088h 4     Bitmask for Supported Languages (3Eh for Europe) (as wifi_flash)
008Ch 4     Unknown (00,00,00,00)  (bit0=flag for 4004020h.bit0=wifi ?)
0090h 1     Console Region (0=JPN, 1=USA, 2=EUR, 3=AUS, 4=CHN, 5=KOR)
0091h 12    Serial/Barcode (ASCII, 11-12 characters; see console sticker)
009Dh 3     Unknown (00,00,3C)                    ;"<"
00A0h 4     Title ID LSBs for Launcher ("PANH", aka HNAP spelled backwards)
00A4h 3F5Ch Unused (FFh-filled)
```

Entries [0088h..009Fh] are copied to [2FFFD68h..2FFFD7Fh]. Entry [00A0h] is used to construct the region-specific filename of the Launcher (System Menu).
The RSA with Console ID means that one cannot change the region/language stuff.
The 3DS has similar data stored in file SecureInfo_A on 3DS partition.

FAT16:\sys\HWINFO_N.dat (16Kbytes)

```
  0000h 14h   SHA1 on entries [088h..09Bh]
  0014h 6Ch   Zerofilled
  0080h 4     Header, Version or so (00000001h)
  0084h 4     Header, Size of entries at [0088h..009Bh] (00000014h)
  0088h 4     Some per-console ID (used what for?)
  008Ch 10h   Some per-console ID (used in "Tad Files")
  009Ch 3F64h Unused (FFh-filled)
```
Entries [0088h..009Bh] are copied to [2000600h..2000613h].

FAT16:\sys\HWID.sgn (256 bytes)
```
  0000h 100h  RSA-OpenPGP-SHA1 across... whatever?
```
Seems to be used only by DSi Shop. The RSA keys are unknown for retail version. Also unknown WHAT the SHA1 is computed on (probably some console and/or region IDs).
The System Updater tool (for debug version) contains both public and private RSA keys for the file; the keys don't work for retail version though.
The OpenPGP bytes are same as for SWI 23h (but with more FFh padding bytes due to the 100h-byte RSA size).

FAT16:\shared2\0000 (2048K) (sound recorder)
Huge 2Mbyte file with several used areas (and many zerofilled areas).
Contains a FAT12 filesystem with several voice .dat files for the Sound Recorder of the Nintendo DSi Sound utility. Unused clusters seem to contain garbage (maybe un-encrypted eMMC sectors).
```
  voice181110082156510000010001.dat  ;14402h bytes
  voice201310182112420000010001.dat  ;14402h bytes
  voice191110082157080000010001.dat  ;14402h bytes
  voice001310182114110031100001.dat  ;14402h bytes
  voice011504181444050021100001.dat  ;14402h bytes
  voiceNNYYMMDDHHMMSS00NN10001.dat  ;14402h bytes
```
Note: The DSi Sound utility is additionally having a 512Kbyte private.sav file (also containing a FAT12 filesystem; although it seems to contain only a MBR, FATs, and an empty Root directory - plus garbage in unused clusters).

FAT16:\import
FAT16:\progress
FAT16:\tmp\es\write
Empty folders.

DSi SD/MMC Firmware Wifi Firmware

FAT16:\title\0003000f\484e4341\content\000000vv.app (aka Wifi Firmware)
WLFIRM aka WLANFIRM (compressed, non-executable datafile, for all regions, v0:13B80h bytes, v1:13BA0h bytes, v2:17E60h bytes). This file contains Wifi Firmwares for the Xtensa CPU in the Atheros AR60xxG chips on the DWM-W0xx Wifi Daughterboards.
```
  DSi Firmware 1.0            --> Wifi Firmware v0 (supports AR6002)
  DSi Firmware 1.1 thru 1.2   --> Unknown (presumably v0 or v1)
  DSi Firmware 1.3            --> Wifi Firmware v1 (supports AR6002)
  DSi Firmware 1.4 thru 1.4.5 --> Wifi Firmware v2 (supports AR6002+AR6013)
  Note: The AR6002 part is exact same in v1 and v2 (with same SHA1 in Part 1)
  However, part 1.c was slightly smaller in v0, apparently some small bugfix.
```
The BIOS ROM in the AR60xxG chips can be extended/upgraded by uploading the firmware into RAM (the RAM isn't nonvolatile, so this must be done each time after power-up, in the DSi this appears to be done by the System Menu (launcher), so DSi games probably don't need to upload the firmware themselves... unless maybe after some kind of reset or power-saving situations?).
```
  00000h 80h    RSA-SHA1    (on [00080h..0009Fh]) (via RSA key from BIOS) ;\
  00080h 14h    Header SHA1 (on [000A0h..000FFh])                         ; SHA
  00094h 4      Header Size (00000060h, for entries 000A0h..000FFh)       ;
  00098h 8      Zerofilled                                                ;/
  000A0h 2      Version (0000h,0001h,0002h for v0,v1,v2)          ;\
```

```
  000A2h 1      Number of parts (01h..02h)   ;(02h in v2 only)    ; Header
  000A3h 1      Unknown/zero?    (00h)                            ;
  000A4h 4      Part 1 Start (00000100h) ;(in v1: E0h)  ;\Part 1  ; with IDs
  000A8h 4      Part 1 Size  (00013AC0h)                ; DWM-W015; as in wifi
  000ACh 4      Part 1 ID    (00000001h) (=DWM-W015)    ; AR6002G ; flash[1FDh]
  000B0h 14h    Part 1 SHA1  (on [00100h..13BBFh])      ;/        ;
  000C4h 4      Part 2 Start (00013BC0h)                ;\Part 2  ; ;\
  000C8h 4      Part 2 Size  (000042A0h)                ; DWM-W024; ; not in
  000CCh 4      Part 2 ID    (00000002h) (=DWM-W024)    ; AR6013G ; ; version 1
  000D0h 14h    Part 2 SHA1  (on [13BC0h..17E5Fh])      ;/        ; ;/
  000E4h 1Ch    Zerofilled (padding to 20h-byte boundary)         ;/
  00100h 1      Part 1 num subheader's    (04h) (a/b/c/d)                   ;\
  00101h 1      Part 1 num ChipID's       (02h)                             ;
  00102h 2      Part 1 offset to ChipID's (0044h)                           ;
  00104h 10h    Part 1.a firm/main (00000080h,00013458h,80000001h,00502400h) ;
  00114h 10h    Part 1.b database  (000134E0h,000002BCh,00000002h,0052D944h) ;
  00124h 10h    Part 1.c stub/code (000137A0h,000002DEh,00000004h,00515000h) ;
  00134h 10h    Part 1.d stub/data (00013A80h,00000030h,00000005h,00502400h) ;
  00144h 8      Part 1 ChipID 1 ;alternate IDs ?       (02010001h,20000188h) ;
  0014Ch 8      Part 1 ChipID 2 ;CHIP_ID, ROM_VERSION  (02000001h,20000188h) ;
  00154h 4      Part 1 Firmware Version: 2.1.0.123              (2100007Bh) ;
  00158h 0Ch    Part 1 RAM vars/base/size    (00500400h,00500000h,0002E000h) ;
  00164h 1Ch    Zerofilled                                                   ;
  00180h 13460h Part 1.a data (13458h compressed bytes, +8 bytes zeropadding);
  135E0h 2C0h   Part 1.b data (2BCh bytes, +04h bytes zeropadding) ;database ;
  138A0h 2E0h   Part 1.c data (2DEh bytes, +02h bytes zeropadding) ;stubcode ;
  13B80h 40h    Part 1.d data (30h bytes, +10h bytes zeropadding)  ;stubdata ;/
  13BC0h 1      Part 2 num subheader's    (04h)                             ;\
  13BC1h 1      Part 2 num ChipID's       (02h)                             ;
  13BC2h 2      Part 2 offset to ChipID's (0044h)                           ;
  13BC4h 10h    Part 2.a firm/main (00000080h,00002EECh,80000001h,00524C00h) ;
  13BD4h 10h    Part 2.b database  (00002F80h,00000FC0h,00000002h,0053F040h) ;
  13BE4h 10h    Part 2.c stub/code (00003F40h,00000312h,00000004h,00527000h) ;
  13BF4h 10h    Part 2.d stub/data (00004260h,00000038h,00000005h,00524C00h) ;
  13C04h 8      Part 2 ChipID 1 ;CHIP_ID, ROM_VERSION  (0D000000h,23000024h) ;
  13C0Ch 8      Part 2 ChipID 2 ;alternate IDs?        (0D000001h,23000024h) ;
  13C14h 4      Part 2 Firmware Version: 2.3.0.108              (2300006Ch) ;
  13C18h 0Ch    Part 2 RAM vars/base/size    (00520000h,00520000h,00020000h) ;
  13C24h 1Ch    Zerofilled                                                   ;
  13C40h 2F00h  Part 2.a data (2EECh compressed bytes,+14h bytes zeropadding);
  16B40h FC0h   Part 2.b data (FC0h bytes, +00h bytes zeropadding)           ;
  17B00h 320h   Part 2.c data (312h bytes, +0Eh bytes zeropadding)           ;
  17E20h 40h    Part 2.d data (38h bytes, +08h bytes zeropadding)            ;/
```

The a/b/c/d subheaders consist of File Source Offset (relative to Start of Part 1/2 accordingly), Length, ID/Flags, and RAM Destination Address.

The stub/code and stub/data parts are loaded and executed first (the stub is reading calibration data from I2C bus EEPROM; this is decoupled from the main firmware because hardware implementations could use different calibration sources like I2C chips or SPI chips or mass-storage devices).

Thereafter, firm/main and database are loaded and executed. The "LZ" compressed firm/main part is automatically decompressed on the Xtensa side, the "LZ" stuff is some kind of "tag,len,disp" format:

```
  Part 1.a data: 9F,FF,FF,FF,FF,FF,FF,00,00,00,00,9F,04,04,..
  Part 2.a data: 5E,00,00,00,00,5E,04,04,5E,08,08,41,5F,49,..
```

The first byte identifies the "tag" value (this should be the value used least often in the uncompressed data). The following bytes are plain uncompressed data, mixed with "tag,len,disp" values (which will copy "len" bytes from "dest-disp" to "dest"). A special case is "tag,00h", which will store the "tag" value at dest. The len and disp values can consist of one or more byte(s) each (the LSB aka last byte is indicated by bit7=0; for example, "84h,86h,0Fh" would mean 01030Fh). For some odd reason, the values are always "len<=disp" (even for zerofilled regions where "len>disp" would be useful).

DSi SD/MMC Firmware System Settings Data Files

The DSi stores System Settings (and Title ID of most recent System Menu index) on eMMC in two identical files: TWLCFG0.dat and TWLCFG1.dat.
If both files are intact then the newer file is taken (as indicated by the update counter; for some weird reason, the DSi System Menu is always updating BOTH files, so they are usually both "newer").
For NDS compatibility, some of the data is additionally stored on Wifi FLASH:
DS Firmware User Settings
The TWL data and the NDS-style data are also copied to RAM:

```
2000400h 128h   TWLCFGn.dat bytes [088h..1AFh]
2FFFC80h 70h    Wifi FLASH User Settings (fmw[newest_user_settings])
2FFFDFCh 4      Pointer to 2000400h
```

For some reason, most DSi games are containing some initialization code for repairing or initializing the above pointer, as so:

```
if [2FFFDFCh]=0 then [2FFFDFCh]=2000400h
```

The RAM data at 2000400h isn't actually used by too many games though (one program that is using it is Flipnote).
Some games are attempting to adopt the System Setting's language selection as game language, games should do that only if they do support that language (a bad example is the german version of Magic Made Fun: Deep Psyche, which defaults to French when using English as system language).

FAT16:\shared1\TWLCFG0.dat (16Kbytes) (System Settings Data)
FAT16:\shared1\TWLCFG1.dat (16Kbytes) (System Settings Data)

```
File RAM  Siz   Description
000h -    14h   SHA1 on entries [088h..1AFh]
014h -    6Ch   Zerofilled
080h -    1     Version or so (01h)
081h -    1     Update Counter (0..7Fh, wraps after 7bit)     ;fmw_user[070h]
082h -    2     Zero (0000h)
084h -    4     Size of below RAM area (00000128h)
088h 000h 1     Unknown (0Fh) (bit3 set when wireless comms are enabled)
089h 001h 2     Zerofilled
08Bh 003h 1     Unknown (01h) (happens to be 00h after.. country change?)
08Ch 004h 1     Zero
08Dh 005h 1     Country code, same as Wii country codes (eg. 40h=Albania)
08Eh 006h 1     Selected Language (eg. 1=English)        ;fmw_user[064h,075h]
08Fh 007h 1     RTC Year (last date change) (max 63h=2099)    ;fmw_user[066h]
090h 008h 4     RTC Offset (difference in seconds on change)  ;fmw_user[068h]
094h 00Ch 4     Zerofilled (or FFh-filled) (=MSBs of above?)
098h 010h 1     Flags (01h) (bit0 set when EULA was accepted) (0=newcountry?)
                ...or EULA version (to be same/higher than carthdr[20Eh])?
099h 011h 9     Zerofilled
0A2h 01Ah 1     Alarm Hour   (0..17h)                         ;fmw_user[052h]
0A3h 01Bh 1     Alarm Minute (0..3Bh)                         ;fmw_user[053h]
0A4h 01Ch 2     Zerofilled
0A6h 01Eh 1     Alarm Enable (0=Off, 1=On)                    ;fmw_user[056h]
0A7h 01Fh 2     Zerofilled
0A9h 021h 4     Unknown (09 1E 00 03)  (2nd.byte.LSB E=English, F=French ??)
0ADh 025h 3     Zerofilled
0B0h 028h 8     Title ID (most recent System Menu selection)      ;cart[230h]
0B8h 030h 2x2   TSC calib (adc.x1,y1) 12bit ADC-position      ;fmw_user[058h]
0BCh 034h 2x1   TSC calib (scr.x1,y1) 8bit pixel-position     ;fmw_user[05Ch]
0BEh 036h 2x2   TSC calib (adc.x2,y2) 12bit ADC-position      ;fmw_user[05Eh]
0C2h 03Ah 2x1   TSC calib (scr.x2,y2) 8bit pixel-position     ;fmw_user[062h]
0C4h 03Ch 4     Unknown (9C 20 01 02)
0C8h 040h 4     Zerofilled
0CCh 044h 1     Favorite color (also Sysmenu Cursor Color)    ;fmw_user[002h]
0CDh 045h 1     Zero
0CEh 046h 2     Birthday (month, day)                   ;fmw_user[003h..004h]
0D0h 048h 14h+2 Nickname (UCS-2), max 10 chars+EOL      ;fmw_user[006h..019h]
0E6h 05Eh 34h+2 Message (UCS-2), max 26 chars+EOL       ;fmw_user[01Ch..04Fh]
11Ch 094h 1     Parental Controls Flags (bit0=Parental, bit1-6=Pictochat,etc)
```

```
  11Dh 095h 6     Zero
  123h 09Bh 1     Parental Controls Region (0=Off, 3=German/USK, 4=French?)
  124h 09Ch 1     Parental Controls Years of Age Rating (00h..14h)  ;cart[2F0h]
  125h 09Dh 1     Parental Controls Secret Question (00h..05h)
  126h 09Eh 1     Parental Controls Unknown (can be 00h, 06h, or 07h)
  127h 09Fh 2     Zero
  129h 0A1h 4+1   Parental Controls PIN (ASCII digits) 4 digits+EOL
  12Eh 0A6h 80h+2 Parental Controls Secret Answer (UCS-2), max 64 chars+EOL
  1B0h -    3E50h Unused (FFh-filled)
```
Additionally, there's some stuff in RAM (maybe current Wifi Firmware version):
```
  -    1E0h 1     WlFirm Type (1=DWM-W015, 2=DWM-W024) (as wifi_flash[1FDh])
  -    1E1h 1     WlFirm Unknown (zero)
  -    1E2h 2     WlFirm CRC16 with initial value FFFFh on [1E4h..1EFh]
  -    1E4h 0Ch   WlFirm Version? RAM_area? (as from "Wifi Firmware" file)
  -    1F0h 10h   WlFirm Unknown (zero)
  -    200h 14h   Hexvalues from HWINFO_N.dat
  -    214h 0Ch   Unused/padding? (zero)
```

Unknown TWLCFG entries:
```
  Language and Flags  2  fmw_user[064h] (particulary: Flags)
  More Parental Control stuff
```
Parental controls fields are all zero when not in use.

DSi SD/MMC Firmware Version Data File

verdata (00030005-HNLx) is a bundle of data which corresponds to a release of the "System Menu" -- every time Nintendo announces a new version of the system menu, they will update one or more other titles and then update this title. The verdata filesize is constant for all versions/regions (1B50h bytes). Existing verdata downloads on Nintendo's server are:
```
  00000001..00000009  jpn         (9 versions)
  00000003..00000009  usa/eur/aus (7 versions)
  00000001..00000006  chn         (6 versions)
  00000002..00000006  kor         (5 versions)
```
Apart from those downloads/updates, each region did probably originally have an older version pre-installed. The japanese 1.0J firmware did oddly have 00000000.app files for BOTH japan and usa installed (whereof the latter contained 0.1A instead of 1.0A as version string).

FAT16:\title\0003000f\484e4cgg\content\000000vv.app (aka Version Data)
```
  0000h 80h  RSA-SHA1 on entries [0080h..end of file]
  0080h ...  NARC (Nitro Archive) ...
```
The NARC is a nintendo-specific virtual filesystem. For details, see:
[DS Cartridge NitroROM and NitroARC File Systems]
The NARC archive contains the following files:
```
 twl-nup-cert.der    - server cert for software update server
 twl-nup-prvkey.der  - client-side private key for software update server
 twl-shop-cert.der   - server cert for Shopping Channel server
 twl-shop-prvkey.der - client-side private key for Shopping Channel server
 NintendoCA-G2.der   - Certificate Authority cert, used to sign the other certs
 eula_url.bin        - URL to the EULA text for this system update,
                       generally https://cfh.t.app.nintendowifi.net/eula/
 nup_host.bin        - server to query for the next system update,
                       generally nus.t.shop.nintendowifi.net:443
 time_stamp.bin      - build date for this version, eg. 00281108 (28 Nov 2008)
 user_area_size.bin  - eg. 08000000h (signed) (=128Mbyte?) (aka 1024 "blocks"?)
 version.bin         - machine and human-readable version numbers for this
                       version of the System Menu, eg.
   0000: 01000300 31002e00 33004500 00000000  ....1...3.E.....
   0010: 00000000 00000000 00000000 00000000  ................
   bytes 0 and 1 are the major version number, bytes 2 and 3 are the minor
```

```
  version number, and the rest of the file is the human-readable
  UCS-2 version number displayed in the Settings menu as the "System Menu
  Version".
```
The four "twl-*.der" files are encrypted with ES Block Encryption (using a fixed key):
```
KEY[00h..0Fh] = Constant (08,2F,61,38,...)  ;from ARM7BIOS
```
DSi ES Block Encryption
Other titles access the NARC by reading from eg. "verdata:/version.bin".

DSi Firmware Versions
```
0.1    31 Jul 2008  Pre-release v0.1A (accidentally included in v1.0J)
1.0    09 Sep 2008  Pre-installed v1.0J version (the actual file in v1.0J)
1.0?   22 Oct 2008? First Update(??) to Japanese Region DSi System Menu
1.1?   ?            dsibrew:NoneSuch?, wikipedia:Preinstalled1stJpnVersion??
1.2    18 Dec 2008  Second Update to Japanese Region DSi System Menu
1.3    03 Apr 2009  Launch Day (USA, EUR, AUS), new "start DSi Camera" button
1.4    29 Jul 2009  Blocks NDS flashcarts, Facebook support to share photos
1.4.1  07 Sep 2010  Blocks more NDS flashcarts
1.4.2  10 May 2011  Blocks DSiWare exploits on SD card (sudokuhax etc.)
1.4.3  29 Jun 2011  Blocks more NDS flashcarts (only whitelist was updated)
1.4.4  21 Mar 2012  Blocks DSi cart exploits (CookingCoach/ClassicWordGames)
1.4.5  11 Dec 2012  Blocks more NDS flashcards
```
JAP region launched first (unknown if there was any pre-installed version prior to the v1.0 update (or if v1.0 was really released as "update" at all), unknown if v1.1 did also exist).
USA/EUR/AUS regions launched on 03 Apr 2009 (so only v1.3 and up exist as update?; but they had an older version pre-installed: v1.2U is known to exist).
CHN region launched on 11 Sep 2010 with firmware v1.4.2C (or more probably with v1.4.1C pre-installed?).
KOR region launched at unknown date (probably near chinese launch date) (korean v1.4.1K is known to exist).
CHN and KOR do have version numbers one step higher than normal regions (ie. v1.4 through v1.4.5 are called v1.4.1 through v1.4.6 in china/korea).

DSi SD/MMC Firmware Nintendo DS Cart Whitelist File

FAT16:\title\0003000f\484e4841\content\00000001.app (aka NDS Cart Whitelist)
The NDS Cart Whitelist contains checksums of all officially released licensed NDS cartridges (newer NDS Carts that aren't included in the list must contain extended NDS Cart Headers with RSA signatures).
That means, unlike the original NDS, the DSi refuses to boot any unlicensed/homebrew NDS software (though some "DSi compatible" FLASH carts are bypassing that restriction via exploits in licensed NDS games).
Below Whitelist example is from firmware v1.4E:
```
Part 1 ("NDHT") is same in v1.0 through v1.4.5:
 00000h 4          ID "NDHT"                                          ;\
 00004h 80h        RSA-SHA1 on [00084h..286A7h]                       ;
 00084h 4          Number of titles (00000D76h) (=3446)               ;
 00088h D76h*30h   Titles (30h bytes each, with two SHA1s)            ;/
Part 2 ("NDHX") is same in v1.4 through v1.4.5 (doesn't exist in v1.3):
 286A8h 4          ID "NDHX"                                          ;\
 286ACh 80h        RSA-SHA1 on [2872Ch..4AFBFh]                       ;
 2872Ch 4          Number of titles (000013BCh) (=5052)               ;
 28730h 13BCh*1Ch  Titles (1Ch bytes each, only one SHA1)             ;/
Part 3 ("NDHI") differs in v1.4 versus v1.4.5 (doesn't exist in v1.3):
 4AFC0h 4          ID "NDHI"                                          ;\
 4AFC4h 80h        RSA-SHA1 on [4B044h..4B1B7h]                       ;
 4B044h 4          Number of titles (04h in v1.4E)    ;60h in v1.4.5E ;
 4B048h 4*5Ch      Specials for A3TE,A6WE,YF7E,YOUF   ;210h on New3DS  ;/
Footer:
 4B1B8h 13         Version String ("2832",0Dh,0Ah,"10619",0Dh,0Ah in v1.4E) ;\
 4B1C5h 11         Random garbage (padding to 10h-byte boundary)            ;/
```
The Version String at the end can be:
```
00000000.app v1.0J   (at 286A8h) "2435",0Ah,"8325",0Ah          ;with LF's
```

```
  00000000.app v1.3U   (at 286A8h) "2435",0Ah,"8325",0Ah          ;with LF's
  00000001.app v1.4E   (at 4B1B8h) "2832",0Dh,0Ah,"10619",0Dh,0Ah ;with CRLF's
  0000000x.app v...    (?)
  00000006.app v1.4.5E (at 4D2C8h) "3067",0Ah,"11437",0Ah         ;with LF's
  0000000x.app v...    (?)
  0000000e.app New3DS  (at 56E08h) "3106",0Ah,"11437",0Ah         ;with LF's
```
BUG: The Whitelist's RSA signatures are NOT checked in firmware v1.4E (whilst other older/newer firmwares like 1.3U, 1.4.2E, and 1.4.5E are checking those signatures).

NDHT Title Structure (30h bytes each):
```
 This contains all NDS titles released prior to DSi firmware v1.0.
  Start Length  Description
  000h  4   Title ID (Gamecode)
  004h  4   Title version
  008h  20  Phase 1 SHA1-HMAC on 160h-byte cartheader and ARM9+ARM7 areas (?)
  01Ch  20  Phase 2 SHA1-HMAC on ARM9 Overlay and NitroFAT (zero if no overlay)
```
NDHX Title Structure (1Ch bytes each):
```
 This contains all NDS titles released prior to DSi firmware v1.4.
  000h  4   Title ID (Gamecode)
  004h  4   Title version
  008h  20  Phase 3 SHA1-HMAC on Icon/Title
```
NDHI Title Structure (5Ch bytes each):
```
 This contains extra checks for detecting hacked/exploited NDS titles.
  000h  4   Title ID (Gamecode)
  004h  4   Title version
  008h  8*8 Offset+Length for up to 8 regions (or 0,0=None)
  048h  20  Phase 4 SHA1-HMAC on above region(s)
```
The 40h-byte SHA1-HMAC keys are contained in Launcher (61h,BDh,DDh,72h,... for Phase 1+2, and 85h,29h,48h,F3h,... for Phase 3+4). The RSA key is also contained in Launcher (C7h,F4h,1Dh,27h,... for all Phases; though the RSA key is missing in firmwares where Nintendo forgot to implement the RSA check, eg. in v1.4E).

Example values for Metroid Demo ("AMFE"):
```
  41 4D 46 45 00 00 00 00                                     ;\
  95 9A B3 09 B7 4E AF 29 2E 97 61 B9 DC E9 5F FE 86 5C 91 4E ; NDHT
  D3 94 43 02 64 3A AF C5 D1 E1 3B C0 47 4A A2 98 AB 5D 71 8F ;/
  41 4D 46 45 00 00 00 00                                     ;\NDHX
  51 24 FE EF D4 3C 22 42 CC 17 13 0A 72 F8 FA 3B 4D 83 2A B1 ;/
```
Specials related to games:
```
  NTR-A3TE-USA = Tak: The Great Juju Challenge
  NTR-A6WE-USA = FIFA World Cup 2006
  NTR-YF7E-USA = Fish Tycoon
  NTR-YOUF-FRA = Samantha Oups!
```

Newer NDS Carts with RSA
NDS games released after DSi firmware v1.0 have RSA headers without Icon SHA1
NDS games released after DSi firmware v1.4 have RSA headers with Icon SHA1
Accordingly, NDS games released between DSi firmware v1.0 and v1.4 do have whitelist NDHX entries (for the icon), but don't need NDHT entries (since that's already covered by the RSA header).
Related carthdr flags are:
```
  cart[1BFh].bit6 = Cart Header RSA Signature exists
  cart[1BFh].bit5 = Cart Header has Icon SHA1 at [33Ch]
  cart[378h]      = SHA1 (same as whitelist Phase 1)
  cart[38Ch]      = SHA1 (same as whitelist Phase 2)
  cart[33Ch]      = SHA1 (same as whitelist Phase 3) (if above bit5=1)
```

DSi SD/MMC Camera Files - Overview

Photos/Frames
Photos can be taken via Nintendo DSi Camera utitilty (or alternately, directly via hotkeys in System Menu; which will be automatically flagging the photos with "Star" stickers, which will cause them to be shown as System Menu background image).
Frames are masks (with transparent pixels) that can be put onto photos. The Frames can be created via Nintendo DSi Camera utility (Camera, select Frame (upper-right Lens option), accept that Lens, then click Create Frame; the procedure then is to take a photo, and to rub-out pixels on touchscreen to make them transparent).

Internal/External Storage
The Camera is storing further data on the eMMC FAT12 partition:
```
FAT12:\photo\DCIM\100NIN02\HNI_nnnn.JPG                      ;camera photos
FAT12:\photo\private\ds\app\484E494A\pit.bin                  ;camera info
FAT12:\photo\private\ds\app\484E494A\DCIM\100NIN02\HNI_nnnn.JPG;camera frames
```
Camera data can be copied to SD card (via Nintendo DSi Camera, Options, Copy):
```
SD:\DCIM\nnnNIN02\HNI_nnnn.JPG                           ;camera photos
SD:\private\ds\app\484E494A\pit.bin                      ;camera info
SD:\private\ds\app\484E494A\DCIM\nnnNIN02\HNI_nnnn.JPG    ;camera frames
```
And, in internal eMMC only (not on SD), the DSi is somewhere storing Calendar entries (some sort of bitmaps with optional handwritten comments drawn via touchscreen).

File/Folder Numbers
The "nnnNIN02" folders are numbered "100NIN02" through "999NIN02". The first folder is usually 100NIN02, unless another "100xxxxx" folder did already exist (eg. if the SD card contains a "100CANON" folder, then DSi would start at 101NIN02 or higher).
The trailing "02" of the "nnnNIN02" folders appears to be fixed for DSi photos (folder name "nnnNIN01" is reserved for Wii screenshots).
The "HNI_nnnn.JPG" filenames are numbered "HNI_0001.JPG" through "HNI_0100.JPG", thereafter, the DSi will switch to next higher folder number, and wrap to using "HNI_0001.JPG" as first filename in that folder.
The weird "484E494A" folder name is based on the japanese Nintendo DSi Camera's gamecode (HNIJ) converted to an 8-digit uppercase HEX string (this appears to be always the japanese gamecode, even on european DSi consoles).

DSi SD/MMC Camera Files - JPEG's

Overall JPEG Format (big-endian)
```
Offs   ID    Len   Data
0000h  FFD8h                                    ;(start of image)    ;SOI
0002h  FFE1h,10C4h,"Exif",00h,00h,<Exif Body>    ;(extra "Exif" data) ;APP1
10C8h  FFC0h,0011h,08h,01E0h,0280h,03h,012100h,021101h,031101h       ;SOF0
10DBh  FFDBh,0084h, 00 06 04 05 06 05 04 06 06 05 06 07 07 .. 28 28 28 ;DQT
1161h  FFC4h,01A2h, 00 00 01 05 01 01 01 .. .. F8 F9 FA              ;DHT
1305h  FFDAh,000Ch,03h,010002h,110311h,003F00h    ;(start of scan)    ;SOS
1313h  E6 76 F4 DD 4F 0A 3B 60 0F 4C D7 9E 9A 93 3D 4B EE 98 B8 .. ..
AB4Fh  FFD9h                                    ;(end of image)      ;EOI
```

Exif Body for Nintendo DSi Photos
The Exif data consists of several headers/footers and data blocks, mixed with several "IFD" tables.
```
Offs  Siz  ExID  Type  Length    Offset    ;<-- Format for "IFD" Tables
0000h 4    "MM",002Ah                     ;Big-Endian (aka Motorola)
0004h 4    00000008h                      ;first IFD offset (IFD0)
```
IFD0 (Main Image):
```
0008h 2    0009h                          ;number of IFD0 entries
000Ah 12   010Fh,0002h,00000009h,0000007Ah ;Maker ("Nintendo",0)
0016h 12   0110h,0002h,0000000Bh,00000084h ;Model ("NintendoDS",0)
0022h 12   011Ah,0005h,00000001h,00000090h ;Resolution X (72 dpi)
```

```
  002Eh 12   011Bh,0005h,00000001h,00000098h ;Resolution Y (72 dpi)
  003Ah 12   0128h,0003h,00000001h,00020000h ;Resolution Unit (2=Inches)
  0046h 12   0131h,0002h,00000005h,000000A0h ;Firmware (Gamecode backwards)
  0052h 12   0132h,0002h,00000014h,000000A6h ;Date/Time Modified
  005Eh 12   0213h,0003h,00000001h,00020000h ;Subsampling (2=datum point)
  006Ah 12   8769h,0004h,00000001h,000000BAh ;Exif SubIFD offset
  0076h 4    000001DEh                       ;next IFD offset (IFD1)
  007Ah 9+1  "Nintendo",00h,00h              ;Maker ("Nintendo",0,0)
  0084h 11+1 "NintendoDS",00h,00h            ;Model ("NintendoDS",0,0)
  0090h      00000048h,00000001h             ;Resolution X (72 dpi)
  0098h      00000048h,00000001h             ;Resolution Y (72 dpi)
  00A0h 5+1  "PINH",00h,00h   ;aka HNIP      ;Firmware (Gamecode backwards)
  00A6h 20   "YYYY:MM:DD HH:MM:SS",00h       ;Date/Time Modified
```

Sub IFD:

```
  00BAh 2    000Ah                           ;number of Sub IFD entries
  00BCh 12   9000h,0007h,00000004h,30323230h ;Exif Version ("0220")
  00C8h 12   9003h,0002h,00000014h,00000138h ;Date/Time Original
  00D4h 12   9004h,0002h,00000014h,0000014Ch ;Date/Time Digitized
  00E0h 12   9101h,0007h,00000004h,01020300h ;Components (Y,Cb,Cr)
  00ECh 12   927Ch,0007h,00000042h,00000160h ;Maker dependent internal data
  00F8h 12   A000h,0007h,00000004h,30313030h ;Flashpix Version ("0100")
  0104h 12   A001h,0003h,00000001h,00010000h ;Color Space (1=Normal=sRGB)
  0110h 12   A002h,0004h,00000001h,00000280h ;Pixel Dimension X (640)
  011Ch 12   A003h,0004h,00000001h,000001E0h ;Pixel Dimension Y (480)
  0128h 12   A005h,0004h,00000001h,000001A2h ;Interoperability IFD (R98)
  0134h 4    00000000h                       ;next IFD offset (none)
  0138h 20   "YYYY:MM:DD HH:MM:SS",00h       ;Date/Time Original
  014Ch 20   "YYYY:MM:DD HH:MM:SS",00h       ;Date/Time Digitized
```

Maker dependent IFD (DSi specific):

```
  0160h 2    0002h                           ;number of IFD entries
  0162h 12   1000h,0007h,0000001Ch,0000017Eh ;DSi Signature (IV+MAC)
  016Eh 12   1001h,0007h,00000008h,0000019Ah ;DSi Whatever Zero (Frame info?)
  017Ah 4    00000000h                       ;next IFD offset (none)
  017Eh 12   2E AB A5 D1 FD A8 .. ..         ;DSi Signature (IV)  ;\
  018Ah 16   xx xx xx xx xx xx .. ..         ;DSi Signature (MAC) ;/
  019Ah 8    0000000000000000h               ;<-- different for Frames
```

Interoperability IFD (R98) (some common/useless stuff for JPEGs):

```
  01A2h 2    0003h                           ;number of IFD entries
  01A4h 12   0001h,0002h,00000004h,52393800h ;Stipulated File ("R98",0)
  01B0h 12   0002h,0007h,00000004h,30313030h ;Whatever ("0100")
  01BCh 12   1000h,0002h,00000012h,000001CCh ;Whatever (JPEG Exif Ver 2.2",0)
  01C8h 4    00000000h                       ;next IFD offset (none)
  01CCh 18   "JPEG Exif Ver 2.2",00h         ;Whatever (JPEG Exif Ver 2.2",0)
```

IFD1 (Thumbnail Image):

```
  01DEh 2    0006h                           ;number of IFD1 entries
  01E0h 12   0103h,0003h,00000001h,00060000h ;Compression (1=JPEG)
  01ECh 12   011Ah,0005h,00000001h,0000022Ch ;Resolution X (72 dpi)
  01F8h 12   011Bh,0005h,00000001h,00000234h ;Resolution Y (72 dpi)
  0204h 12   0128h,0003h,00000001h,00020000h ;Resolution Unit (2=Inches)
  0210h 12   0201h,0004h,00000001h,0000023Ch ;Jpeg Offset
  021Ch 12   0202h,0004h,00000001h,00000xxxh ;Jpeg Size (eg. E80h)
  0228h 4    00000000h                       ;next IFD offset (none)
  022Ch 8    00000048h,00000001h             ;Resolution X (72 dpi)
  0234h 8    00000048h,00000001h             ;Resolution Y (72 dpi)
```

Thumbnail Data (160x120pix, in JPEG Format):

```
  023Ch 2    FFD8h                           ;(start of thumbnail/image) ;SOI
  023Eh 13h  FFC0h,0011h,08h,0078h,00A0h,03h,012100h,021101h,031101h     ;SOF0
  0251h 86h  FFDBh,0084h, 00 0A 07 07 08 07 .. ..                        ;DQT
  02D7h 1A4h FFC4h,01A2h, 00 00 01 05 01 .. .. F8 F9 FA                  ;DHT
  047Bh 0Eh  FFDAh,000Ch,03h,010002h,110311h,003F00h   ;(start of scan)  ;SOS
  0489h ...  CC 55 14 F0 3D 2B 8B 4B 9D C2 E3 BD 18 A5 B0 09 B6 .. ..
  10xxh 2    FFD9h                           ;(end of thumbnail/image)   ;EOI
```

The above european gamecode entry differs for other regions. Above offsets are usually as so for DSi jpeg's, but they might change if Nintendo adds/removes some entries, or changes size of some entries (for example, the Model string is said to be different for 3DS, and "Frames" are including a bigger entry, as described below).

DSi Signature (IV+MAC)
The 1Ch-byte Signature is split into a 0Ch-byte IV value (this might be just a random number?), and a 10h-byte MAC value. The MAC is computed via AES-CCM:

```
IV[00h..0Bh]  = First 0Ch-bytes of signature
KEY[00h..0Fh] = Constant (70,88,52,06,...)   ;from BIOS ROM
Zerofill the 1Ch-byte signature area in the JPEG file
Probably zeropad(?) the JPEG file (if filesize isn't a multiple of 16 bytes)
Pass the whole JPEG as "extra associated data" to the AES-CCM hardware
Copy the IV value and computed MAC value back to the JPEG's signature area
```

Unknown if the IV value is just random, and unknown if there are further requirements (such like using same Maker/Model strings or same resolution as in original DSi files).
Locating "ldr rx,=927Ch" opcodes at various locations in Nintendo DSi Camera is easy; but the stuff is handled via numerous sub-functions, including IPC stuff with both ARM7 and ARM9 envolved; which isn't too easy to disassemble.

Exif Body for Nintendo DSi Frames
Frames are using same format above, but with the 8-byte zero entry at [019Ah] replaced by a bigger 8FCh-byte entry (accordingly, the size in IFD entry [016Eh] is also adjusted, and offsets for entries [01A2h..10xxh] are moved up).
The actual change is that 0000000000000000h is changed from 8 byte size to 8FCh-byte size, containing 0000000000000001h, followed by 8F4h extra bytes (with unknown content; maybe the frame mask for transparent pixels; the data doesn't really look like a mask though, unless it's compressed, but then the fixed size would be strange).

IFD Type Values (and Length/Offset)

```
0001h = 8bit Unsigned
0002h = 7bit ASCII
0003h = 16bit Unsigned
0004h = 32bit Unsigned
0005h = 64bit Unsigned Rational (32bit numerator, plus 32bit denominator)
0006h = Reserved
0007h = 8bit General Purpose
0009h = 32bit Signed
000Ah = 64bit Signed Rational (32bit numerator, plus 32bit denominator)
000Bh..FFFFh = Reserved
```

The "Length" value indicates the number of type units, eg. type=16bit, length=3 would mean 6 bytes. If the information fits into 4 bytes then it's stored directly in the 4-byte "Offset" field, otherwise "Offset" is a pointer to the actual information.

DSi SD/MMC Camera Files - pit.bin

FAT12:\photo\private\ds\app\484E494A\pit.bin (8K) (camera info)
SD:\private\ds\app\484E494A\pit.bin (47K) (camera info)

```
0000h       8  ID ("0TIP00_1") (maybe meant to read as PIT01_00 or so)
0008h       2  Number of pit.bin entries (3000 for SD Card) (500 for eMMC)
000Ah       2  Unknown (0001h)
000Ch       2  Next Photo Folder-Number minus 100  (xxxNIN02)
000Eh       2  Next Photo File-Number minus 1      (HNB_0xxx.JPG)
0010h       2  Next Frame Folder-Number minus 100  (xxxNIN02)
0012h       2  Next Frame File-Number minus 1      (HNB_0xxx.JPG)
0014h       2  CRC16 of whole file (with initial value 0000h, and with
               entry [0014h] being treated as 0000h for calculation)
0016h       2  Size of Header (0018h)
```

```
  0018h+N*10h 4  Entry N, Time/Date (seconds since 01 Jan 2000)
  001Ch+N*10h 8  Entry N, Unknown (zerofilled)
  0024h+N*10h 4  Entry N, Flags (see below)
                  0     Used Entry Flag (0=Unused/Deleted, 1=Used)
                  1-10  Folder-Number minus 100 (xxxNIN02)
                  11-17 File-Number minus 1     (0..99 = HNB_0001..0100.JPG)
                  18-19 Sticker (0=None, 1=Star, 2=Clover, 3=Heart)
                  20-21 Type (0,3=Photo, 1=Frame, 2=?)
                  22-23 Unknown (0,2=Normal?, 1=?, 3=Error)
                  24-31 Unused (zero)
  xxx8h       8  Padding for 16-byte filesize alignment (zerofilled)
```

The "Next Photo/Frame" entries contain File/Folder Numbers where the next images will be saved; that file numbers increase after saving, and do eventually wrap to next higher folder number.
The Nintendo DSi Camera utility shows only photos listed in "pit.bin", when manually copying jpg's to SD Card one could:

```
 - Delete "pit.bin" (it'll be recreated with ALL jpgs, sticker flags are lost)
 - Replace an existing 'listed' file by a new file with same filename
 - Manually edit "pit.bin" and adjust its CRC16 checksum
```

Photos are region free, can be viewed from any other DSi's (as long as they are listed in pit.bin). However, they do require some signature in Exif header, so in general, the DSi accepts only images that come from DSi consoles; not images from other sources.

Stickers
Photos with "Star" sticker are shown as background picture in System Menu; this works only for images stored in internal eMMC memory (images on SD Card can have stickers, too, but they are ignored by System Menu).

DSi SD/MMC Flipnote Files

FAT16:\title\00030004\4b4755gg\data\public.sav
This .sav file contains a FAT12 filesystem with following files:

```
  Flipnote(public.sav):\eula.txt     ;128 Kbytes, 20000h - zerofilled
  Flipnote(public.sav):\option.bin   ;256 bytes, 100h    - options
  Flipnote(public.sav):\mark0.pls    ;8000 bytes, 1F40h  - Heart sticker
  Flipnote(public.sav):\mark1.pls    ;8000 bytes, 1F40h  - Crown sticker
  Flipnote(public.sav):\mark2.pls    ;8000 bytes, 1F40h  - Music sticker
  Flipnote(public.sav):\mark3.pls    ;8000 bytes, 1F40h  - Skull sticker
  Flipnote(public.sav):\recent10.pls ;4000 bytes, FA0h   - Recently saved
  Flipnote(public.sav):\friend.pls   ;28800 bytes, 7080h - F7,A0,CD,zeroes..
  Flipnote(public.sav):\remind.pls   ;10240 bytes, 2800h - F7,A0,CD,zeroes..
  Flipnote(public.sav):\latest1.pls  ;256 bytes, 100h    - xxxxxxxx,zeroes..
  Flipnote(public.sav):\nand.pls     ;160000 bytes, 27100h - All files
  Flipnote(public.sav):\ugo\0NN\XNNNNN_NNNNNNNNNNNNN_NNN.ppm - flipnotes
```

Flipnote(public.sav):\option.bin (256 bytes, 100h)

```
  0000h 1   Unknown (02h)        (?)
  0001h 1   Stylus               (00h=Right Hand, 01h=Left Hand)
  0002h 1   Sound Effects        (00h=On, 01h=Off)
  0003h 1   Unknown (01h)        (?)
  0004h 1   Unknown (00h)        (?)
  0005h 1   Unknown (01h)        (?)
  0006h 1   Unknown (01h)        (?)
  0007h 1   Unknown (03h)        (?)
  0008h 1   Unknown (02h)        (?)
  0009h 1   Unknown (01h)        (?)
  000Ah 1   Unknown (00h)        (?)
  000Bh 1   Unknown (01h)        (?)
  000Ch 1   Advanced Tools       (00h=Off, 01h=On)
  000Dh 1   Pages to Trace       (01h..04h=1..4)
  000Eh 1   Frog Display         (01h=Off, 01h=On)
```

```
  000Fh 1   Start on Calendar    (00h=Off, 01h=On)
  0010h 8   Flipnote Studio ID   (64bit User ID) (fixed)
  0018h 2   Checksum             (see below)
  001Ah 2   Date of Birth, Year  (076Ch..0840h=1900..2112)
  001Bh 1   Date of Birth, Month (01h..0Ch=1..12)
  001Ch 1   Date of Birth, Day   (01h..1Fh=1..31)
  001Eh E2h Unknown/unused       (zerofilled)
```
Checksum is computed as "chk=0000h, halfword[18h]=0000h, for i=0 to FFh, chk=chk+(byte[i] xor i), next i, halfword[18h]=chk".

SD:\private\ds\app\4B4755GG\mark0.pls (Heart sticker), 8000 bytes (1F40h)
SD:\private\ds\app\4B4755GG\mark1.pls (Crown sticker), 8000 bytes (1F40h)
SD:\private\ds\app\4B4755GG\mark2.pls (Music sticker), 8000 bytes (1F40h)
SD:\private\ds\app\4B4755GG\mark3.pls (Skull sticker), 8000 bytes (1F40h)
SD:\private\ds\app\4B4755GG\recent10.pls (Recently saved), 4000 bytes (FA0h)
Recently saved files list, and four files with flipnotes that have "stickers" assigned to them (which can be done when clicking "Details" in the flipnote file menu).
```
  0000h   N*3Fh       List of filenames (if any)              ;\encrypted
  N*3Fh+0 1           End of filename list (00h)              ;/
  N*3Fh+1 2           Crippled MD5 checksum bytes [6,8]       ;\unencrypted
  N*3Fh+3 SIZ-N*3Fh-3 Padding to end of file (zerofilled)     ;/
```
The filenames are formatted as so (consisting of device, path, and name):
```
  "sdmc:/private/ds/app/4B4755GG/001/XNNNNN_NNNNNNNNNNNN_NNN.ppm",0Ah
```
The files are encrypted using a fixed XOR pattern (repeated every 40h bytes):
```
  F7h,4Ch,6Ah,3Ah,FBh,82h,A6h,37h,6Eh,11h,38h,CFh,A0h,DDh,85h,C0h
  C7h,9Bh,C4h,D8h,DDh,28h,8Ah,87h,53h,20h,EEh,E0h,0Bh,EBh,43h,A0h
  DBh,55h,0Fh,75h,36h,37h,EBh,35h,6Ah,34h,7Fh,B5h,0Fh,99h,F7h,EFh
  43h,25h,CEh,A0h,29h,46h,D9h,D4h,4Dh,BBh,04h,66h,68h,08h,F1h,F8h
```
The checksum is derived by computing the MD5 checksum across the data bytes (in unencrypted form), and then crippling the 16-byte MD5 value to 2 bytes.
For example, an empty file contains only three bytes: F7h,A0h,CDh (plus zero padding).

SD:\private\ds\app\4B4755GG\001\dirmemo2.lst (files), 160000 bytes (27100h)
```
  0000h   N*1Dh       List of filenames (if any)              ;\encrypted
  N*1Dh+0 1           End of filename list (00h)              ;/
  N*1Dh+1 2           MD5 checksum bytes [6,8]                ;\unencrypted
  N*1Dh+3 SIZ-N*1Dh-3 Padding to end of file (zerofilled)     ;/
```
The filenames are formatted as so (raw names, without path):
```
  "XNNNNN_NNNNNNNNNNNN_NNN.ppm",0Ah
```
Checksum and encryption is same as for the ".pls" files, however, the ".lst" file contains only raw filenames (without path's).

SD:\private\ds\app\4B4755GG\gif\XNNNNN_NNNNNNNNNNNN_NNN.gif
Flipnotes exported to GIF format. Supported are animated GIFs, and separate GIFs for each frame. There appears to be no way to view GIFs, or to convert them back to PPM format.

SD:\private\ds\app\4B4755GG\001\XNNNNN_NNNNNNNNNNNN_NNN.ppm ;normal
SD:\private\ds\app\4B4755GG\YYYYMMDD\NNN\XNNNNN_NNNNNNNNNNNN_NNN.ppm ;backup
Flipnotes animation files.
```
  0000h  4       File ID ("PARA")
  0004h  4       Size of Animation Data (vid)
  0008h  4       Size of Audio Data     (aud) (0=none)
  000Ch  2       Number of Frames minus 1    (NF-1)
  000Eh  2       Unknown (always 24h,00h)
  0010h  2       Lock Flag (0=Open, 1=Locked, prevent editing)
  0012h  2       Preview frame number
  0014h  22      Nickname of Original Author (UCS-2)  ;\max 10 characters
  002Ah  22      Nickname of Last Editor     (UCS-2)  ; (plus ending zero)
  0040h  22      Nickname of User (?)        (UCS-2)  ;/
```

```
  0056h  8       User ID of Original Author  (Flipnote Studio ID)
  005Eh  8       User ID of Last Editor      (Flipnote Studio ID)
  0066h  18      Filename of Original File   (3xHEX, 13xASCII, 2xVER)
  0078h  18      Filename of Current File    (3xHEX, 13xASCII, 2xVER)
  008Ah  8       User ID of Previous Editor  (Flipnote Studio ID)
  0092h  8       Filename Fragment           (3xHEX, 5xHEX)
  009Ah  4       Time/Date (seconds since 1st Jan 2000)
  009Eh  2       Zerofilled
  00A0h  600h    Preview Bitmap (8x6 tiles, aka 64x48 pixels, 4bpp)
  06A0h  2       Size of Animation Table (4*NF)
  06A2h  4       Zerofilled
  06A6h  2       Flags (bit0=Can be set?, bit1=Loop/Repeat, bit6=Set?)
  06A8h  4*NF    Animation Table (offsets in Animation Data for each frame)
  ...    (vid)   Animation Data Frame(s)
  ...    1*NF    Audio Flags for each Frame (bit0-2: Effect 1-3, bit3-7: Zero)
  ...    ..      Padding (0..3 bytes zerofilled, for alignment of next entry)
  ...    4       Size of Background music in bytes (0=not used/empty)
  ...    4       Size of Sound effect #1 in bytes (0=not used/empty, max 2000h)
  ...    4       Size of Sound effect #2 in bytes (0=not used/empty, max 2000h)
  ...    4       Size of Sound effect #3 in bytes (0=not used/empty, max 2000h)
  ...    1       Framespeed for playback          (1..8) aka "8 minus N"
  ...    1       Framespeed when BGM was recorded (1..8) aka "8-decimal"
  ...    14      Zerofilled
  ...    (aud)   Audio Data (BGM, followed by Effects 1, 2, 3) (if any)
  ...    80h     RSA-OpenPGP-SHA1 across all preceeding bytes
  ...    10h     Zerofilled
```

The RSA signature is in OpenPGP SHA1 format (as used by SWI 23h, however Flipnote is using it's own RSA functions instead of the BIOS SWIs). The RSA public/private keys are contained in the Flipnote executable (in the modcrypted area).

Animation Data Frame(s)

```
  Start  Length  Description
  0000h  1       Pen and Paper information
  0001h  48      Layer 1 Line Encoding (48 bytes = 2bit per 192 lines)
  0031h  48      Layer 2 Line Encoding (48 bytes = 2bit per 192 lines)
  0061h  ...     Frame Data for Layer 1
  ...    ...     Frame Data for Layer 2
```

The pen and paper byte at the start is encoded as follows:

```
  0    Paper     (0=Black, 1=White)
  1-2  Layer 1   (0=None, 1=Inverse of Paper, 2=Red, 3=Blue)
  3-4  Layer 2   (0=None, 1=Inverse of Paper, 2=Red, 3=Blue)
  5-6  Unknown
  7    New Frame (0=Change between last frame, 1=Totally new frame)
```

The Line Encoding contains 2bit values for all 192 scanlines (starting with bit0-1 of the first byte; for the top-most(?) scanline). The meaning of the 2bit values is:

```
  0 = Skip Line    (0 bytes)   (0 pixels)
  1 = Packed Line  (4+N bytes) (32bit flags, plus Nx8 pixels)
  2 = Inverse Line (4+N bytes) (32bit flags, plus Nx8 inverted pixels)
  3 = Raw Line     (32 bytes)  (256 pixels)
```

The packed lines contain a 32bit header (with flags for each 8-pixel fragment of the line, bit31 being the left-most fragment), followed by data bytes for each flagged fragment (with 8 pixels per fragment, bit0 being the left-most pixel). The Inverse lines have the same 32bit flags, but for whatever reason, the following data byte(s) are to be XORed with FFh.

Audio Data

```
  First comes the BGM (if used)
  Then comes sound effect #1 (if used)
  Then comes sound effect #2 (if used)
  Then comes sound effect #3 (if used)
```

The sound data seems to be a variant of VOX ADPCM at around 8KHz.

IDs and Filenames
Half of the Flipnote Studio ID is same as the last 4 bytes of the wifi MAC address. The filenames are also containing the last 3 bytes of the MAC address. The "XNNNNN_NNNNNNNNNNNNN_NNN.ppm" filename is also encoded in three header entries:

```
Header  = Filename      = Meaning
3xHEX   = XNNNNN        = Based on MAC address (the "X" in "XNNNNN" is what?)
13xASCII= NNNNNNNNNNNNN = Some 13-digit random number or so as ASCII string
5xHEX   = NNNNNNNNNN    = First 10-digits of above 13-digit string
2xVER   = NNN.ppm       = Trailing version(?) number (hex, decimal?)
```

For the file name "G35B20_0909841CDBEB1_002.ppm":

```
                  <-hex--> <-----ascii----------------> <-asc--> <-n->
Filename (ori)   D3 5B 20 30 39 30 39 38 34 31 43 44 42 45 42 31 00 00
Filename (curr)  D3 5B 20 30 39 30 39 38 34 31 43 44 42 45 42 31 02 00
                  <-hex--> <--hex------->
Filename (frag)  D3 5B 20 09 09 84 1C DB
```

Preview Palette (fixed)
Fullscreen flipnotes can have only four colors: Black, White, Red, Blue). Additional shades like gray, magenta, or dark/light colors exist only in resampled preview images. The Preview List uses pale colors, with higher contrast for the Selected icon.

```
Color  Purpose    (appearance)  Preview   Selected
00h    N/A       (transparent)  1F-1F-1F  1F-1F-1F
01h    Black       (dark grey)  13-13-13  0A-0A-0A
02h    White           (white)  1F-1F-1F  1F-1F-1F
03h    White+Black      (grey)  19-19-19  13-13-13
04h    Red               (red)  1F-12-12  1F-09-09
05h    Red+Black    (dark red)  1B-13-13  18-0A-0A
06h    Red+White         (pink)  1F-19-19  1F-15-15
07h    N/A              (green)  0E-1F-0E  02-1F-02
08h    Blue              (blue)  12-12-1F  09-09-1F
09h    Blue+Black  (dark blue)  13-13-1A  0A-0A-16
0Ah    Blue+White (light blue)  19-19-1F  15-15-1F
0Bh    N/A              (green)  0E-1F-0E  02-1F-02
0Ch    Red+Blue       (magenta)  1A-13-1A  16-0B-16
0Dh    N/A              (green)  0E-1F-0E  02-1F-02
0Eh    N/A              (green)  0E-1F-0E  02-1F-02
0Fh    N/A              (green)  0E-1F-0E  02-1F-02
```

FAT12:\photo\DCIM\100NIN02\HNI_nnnn.JPG ;camera photos
Flipnote doesn't contain an own camera function. However, the drawing utility does allow to import JPGs from the camera partition (ie. images that have been previously taken via the DSi Camera utility).

DSi Atheros Wifi SDIO Interface

AR6002 SDIO Registers
DSi Atheros Wifi SDIO Function 0 Register Summary
DSi Atheros Wifi SDIO Function 1 Register Summary
DSi Atheros Wifi - SDIO Function 1 I/O - mbox_wlan_host_reg
DSi Atheros Wifi Misc
For general info about SDIO protocol and I/O ports, and SDIO Function 0, see
DSi SD/MMC Protocol and I/O Ports

Transfer Protocol (Commands/Events)
DSi Atheros Wifi - Command Summary
DSi Atheros Wifi - Response Summary
DSi Atheros Wifi - Host Interest Area in RAM
DSi Atheros Wifi - BMI Bootloader Commands

DSi Atheros Wifi - MBOX Transfer Headers
DSi Atheros Wifi - WMI Misc Commands
DSi Atheros Wifi - WMI Misc Events
DSi Atheros Wifi - WMI Connect Functions
DSi Atheros Wifi - WMI Channel and Cipher Functions
DSi Atheros Wifi - WMI Scan Functions
DSi Atheros Wifi - WMI Bit Rate Functions
DSi Atheros Wifi - WMI Threshold Functions
DSi Atheros Wifi - WMI Error, Retry and Debug Functions
DSi Atheros Wifi - WMI Priority Stream Functions
DSi Atheros Wifi - WMI Roam Functions
DSi Atheros Wifi - WMI Power Functions
DSi Atheros Wifi - WMI Statistics Function
DSi Atheros Wifi - WMI Bluetooth Coexistence (older AR6002)
DSi Atheros Wifi - WMI Wake on Wireless (WOW) Functions
DSi Atheros Wifi - WMI General Purpose I/O (GPIO) Functions
Additional WMI Functions (NOT implemented in DSi with AR6002, but maybe exist in DSi/3DS with AR6013/AR6014):
DSi Atheros Wifi - Unimplemented WMI Misc Functions
DSi Atheros Wifi - Unimplemented WMI Bluetooth Coexistence (newer AR6002)
DSi Atheros Wifi - Unimplemented WMI Bluetooth Coexistence (AR6003)
DSi Atheros Wifi - Unimplemented WMI DataSet Functions
DSi Atheros Wifi - Unimplemented WMI AP Mode Functions (exists on 3DS)
DSi Atheros Wifi - Unimplemented WMI DFS Functions
DSi Atheros Wifi - Unimplemented WMI P2P Functions
DSi Atheros Wifi - Unimplemented WMI WAC Functions
DSi Atheros Wifi - Unimplemented WMI RF Kill and Store/Recall Functions
DSi Atheros Wifi - Unimplemented WMI THIN Functions
DSi Atheros Wifi - Unimplemented WMI Pyxis Functions
Aside from WMI Commands/Events, it should be obviously also possible to transfer actual data packets, but unknown to do that... maybe it's done through MBOX0 too, and maybe related to WMI_DATA_HDR, WMI_TX_META_V0..3, WMI_RX_META_V0..2 in "wmi.h" and/or to stuff in "htc.h".

DSi Atheros Wifi I2C EEPROM

DSi Atheros Wifi SDIO Function 0 Register Summary

Atheros SDIO "Function 0" area

```
  0:00000 2      Revision       (0011h = CCCRv1.10, SDIOv1.10, SDv1.01)  ;\
  0:00002 2      Function       (0202h = Function 1 enabled/ready)       ;
  0:00004 2      Interrupt Flags(0000h = None enabled/pending)           ;
  0:00006 1      Abort/Reset    (00h)                                    ;
  0:00007 1      Bus Interface  (82h = 4bit mode, pulldown=off)          ;
  0:00008 1      Card Capability(17h)                                    ; CCCR
  0:00009 3      CIS0 Pointer   (001000h = CIS0 at 0:01000h)             ;
  0:0000C ..       ..suspend..? (zero-filled)                            ;
  0:00010 2      Block Size     (0000h = Function 0 Block Size, variable);
  0:00012 1      Power Control  (03h = supports/uses more than 720mW)    ;
  0:00013 2      Bus Speed      (0000h = Supports only SDR12)            ;
  0:00015 1      Driver Strength(00h)                                    ;
  0:00016 1      Interrupt Ext  (00h = No aysnc IRQ support in 4bit mode);
  0:00017 E9h    Reserved       (zero-filled)                            ;/
  0:00100 2      Interface Type (0000h=Not SDIO standard, no CSA)        ;\
  0:00102 1      Power          (00h=No power selection)                 ;
  0:00103 6      Reserved       (zero-filled)                            ;
```

```
0:00109 3      CIS1 Pointer   (001100h = CIS1 at 0:01100h)            ; FBR1
0:0010C 4      CSA Stuff      (zero-filled, CSA isn't supported)      ;
0:00110 2      Block Size     (0080h = Function 1 Block Size, variable);
0:00112 EEh    Reserved       (zero-filled)                           ;/
0:00200 600h   FBR2..FBR7     (zero-filled)                           ;-FBRn
0:00800 800h   Reserved       (zero-filled)                           ;-N/A
0:01000         01 03 D9 01 FF    ;DEVICE (D9h=FUNCSPEC,01h=Siz,FFh=End;\
0:01005         20 04 71 02 00 02 ;MANFID (0271h=Atheros, 0200h=AR6002);
0:0100B         21 02 0C 00       ;FUNCID (0Ch,00h=Standard for SDIO)  ;
0:0100F         22 04 00 00 08 32 ;FUNCE (0800h=MaxBlkSiz,32h=25Mbit/s); CIS0
0:01015         1A 05 01 01 00 02 07            ;\       ;CONFIG       ;
0:0101C         1B 08 C1 41 30 30 FF FF 32 00   ; PROM?  ;CFTABLE_ENTRY;
0:01026         14 00                           ; RAM?   ;NO_LINK      ;
0:01028..01044  FF-filled (1Dh bytes) ;uh?      ;/       ;END          ;
0:01045..010FF  00-filled (BBh bytes) ;unused                           ;/
0:01100         20 04 71 02 00 02 ;MANFID (0271h=Atheros, 0200h=AR6002);\
0:01106         21 02 0C 00     ;FUNCID (0Ch,00h=Standard for SDIO)    ;
0:0110A         22 2A 01        ;FUNCE                                 ;
0:0110D          01 11          ;FUNCE WakeUpSupport(01h), v1.1(11h)   ;
0:0110F          00 00 00 00    ;FUNCE Serial Number (00000000h=None)  ;
0:01113          00 00 00 00 00 ;FUNCE CSA Stuff (00000000h,00h=None)  ;
0:01118          00 08          ;FUNCE Max Block Size (0800h)          ; CIS1
0:0111A          00 00 FF 80    ;FUNCE OCR (80FF0000h)                 ;
0:0111E          00 00 00       ;FUNCE Operate Min/Avg/Max (00,00,00)  ;
0:01121          00 01 0A       ;FUNCE Standby Min/Avg/Max (00,01,0A)  ;
0:01124          00 00 00 00    ;FUNCE Bandwidth Min/Opt (0000h,0000h) ;
0:01128          00 00          ;FUNCE Timeout Enable-till-Rdy (0000h) ;
0:0112A          00 00 00 00    ;FUNCE Operation  Avg/Max (0000h,0000h);
0:0112E          00 01 00 01    ;FUNCE HighCurrentAvg/Max (0100h,0100h);
0:01132          00 01 00 01    ;FUNCE LowCurrent Avg/Max (0100h,0100h);
0:01136         80 01 06        ;VENDOR                                ;
0:01139         81 01 07        ;VENDOR                                ;
0:0113C         82 01 DF        ;VENDOR                                ;
0:0113F         FF              ;END                                   ;
0:01140         01              ;Garbage?                              ;
0:01141..011FF  00-filled (BFh bytes) ;unused                          ;/
0:01200..02FFF  mirrors of 01000h..011FFh (CIS0 and CIS1) (1E00h bytes);\N/A
0:03000..       00-filled (.... bytes) ;unused... reserved             ;/
```

Briefly

```
0:00000         11 00 02 02 00 00 00 82  17 00 10 00 00 00 00 00  ;\
0:00010         00 00 03 00 00 00 00                              ;
0:00017..000FF  unused (zerofilled)                               ;/
0:00100         00 00 00 00 00 00 00 00  00 00 11 00 00 00 00 00  ;\
0:00110         80 00                                             ;
0:00112..00FFF  unused (zerofilled)                               ;/
0:01000         01 03 D9 01 FF                          ;\
0:01005         20 04 71 02 00 02       ;hif.h: 271h    ;
0:0100B         21 02 0C 00                             ;
0:0100F         22 04 00 00 08 32                       ;
0:01015         1A 05 01 01 00 02 07            ;\      ;
0:0101C         1B 08 C1 41 30 30 FF FF 32 00   ; PROM? ;
0:01026         14 00                           ; RAM?  ;
0:01028..01044  FF-filled (1Dh bytes) uh?       ;/      ;
0:01045..010FF  00-filled                               ;/
0:01100         20 04 71 02 00 02                       ;\
0:01106         21 02 0C 00                             ;
0:0110A         22 2A 01 01 11 00  00 00 00 00  00 00 00 00  00 08 00 00
0:0111C               FF 80 00 00  00 00 01 0A  00 00 00 00  00 00 00 00
0:0112C               00 00 00 01  00 01 00 01  00 01
0:01136         80 01 06                                ;
0:01139         81 01 07                                ;
0:0113C         82 01 DF                                ;
0:0113F         FF 01 00                                ;
```

```
  0:01142..011FF  00-filled                                ;/
  0:01200..02FFF  mirrors of 01000..011FF (common cis and function 1 cis) ?
  0:03000..       00-filled
```

DSi Atheros Wifi SDIO Function 1 Register Summary

Atheros SDIO "Function 1" area

```
  1:00000..000FF  Mbox0 (100h bytes) <--DMA----------> Internal 256MB
  1:00100..001FF  Mbox1 (100h bytes) <--DMA----------> Internal 256MB
  1:00200..002FF  Mbox2 (100h bytes) <--DMA----------> Internal 256MB
  1:00300..003FF  Mbox3 (100h bytes) <--DMA----------> Internal 256MB
  1:00400..005FF  Control Registers  <--WINDOW_DATA--> Internal 256MB
  1:00600..007FF  CIS Window; Window ---huh???-------> Internal 256MB ???
  1:00800..00FFF  Mbox0 Alias (bigger 800h bytes alias)
  1:01000..017FF  Mbox1 Alias (bigger 800h bytes alias)
  1:01800..01FFF  Mbox2 Alias (bigger 800h bytes alias)
  1:02000..027FF  Mbox3 Alias (bigger 800h bytes alias)
  1:02800..03FFF  Extra Mbox0 Alias "for future usage" (1800h bytes)
  1:04000..1FFFF  Unspecified
```

mbox_wlan_host_reg.h -- in SDIO Function 1 address space

```
  1:00000h 100h    Mbox0 (100h bytes) <--DMA----------> Internal 256MB
  1:00100h 100h    Mbox1 (100h bytes) <--DMA----------> Internal 256MB
  1:00200h 100h    Mbox2 (100h bytes) <--DMA----------> Internal 256MB
  1:00300h 100h    Mbox3 (100h bytes) <--DMA----------> Internal 256MB
  1:00400h 1       HOST_INT_STATUS    (R)
  1:00401h 1       CPU_INT_STATUS     (R/W)
  1:00402h 1       ERROR_INT_STATUS   (R/W)
  1:00403h 1       COUNTER_INT_STATUS (R)
  1:00404h 1       MBOX_FRAME         (R)
  1:00405h 1       RX_LOOKAHEAD_VALID (R)
  1:00406h 1       HOST_INT_STATUS2             ;\GMBOX related, hw4/hw6 only
  1:00407h 1       GMBOX_RX_AVAIL               ;/
  1:00408h 1x4     RX_LOOKAHEAD0[0..3] (R)
  1:0040Ch 1x4     RX_LOOKAHEAD1[0..3] (R)
  1:00410h 1x4     RX_LOOKAHEAD2[0..3] (R)
  1:00414h 1x4     RX_LOOKAHEAD3[0..3] (R)
  1:00418h 1       (HOST_)INT_STATUS_ENABLE  (R/W)
  1:00419h 1       CPU_INT_STATUS_ENABLE     (R/W)
  1:0041Ah 1       ERROR_(INT_)STATUS_ENABLE (R/W)
  1:0041Bh 1       COUNTER_INT_STATUS_ENABLE (R/W)
  1:0041Ch 1x4     PAD1                 (FFh,6Eh,D7h,BFh - maybe some mirror?)
  1:00420h 1x8     COUNT[0..7]         (R/W)
  1:00428h 1x24    PAD2
    00428h 4       -   (mirror of 1:00468h?)
    0042Ch 4       -   (mirror of 1:0041Ch?)
    00430h 4       -   (mirror of 1:00410h?)
    00434h 4       -   (mirror of 1:00...h?)
    00438h 4       -   (mirror of 1:00468h?)
    0043Ch 4       -   (mirror of 1:0041Ch?)
  1:00440h 4x8     COUNT_DEC[0..7]     (R, or Write=any)
  1:00460h 1x8     SCRATCH[0..7]       (R/W)
  1:00468h 1       FIFO_TIMEOUT        (R/W)
  1:00469h 1       FIFO_TIMEOUT_ENABLE (R/W)
  1:0046Ah 1       DISABLE_SLEEP       (R/W)
  1:0046Bh 1x3     -
  1:0046Eh 1       LOCAL_BUS_ENDIAN    (R/W) (AR6001 only, not hw2/hw4/hw6)
  1:0046Fh 1       -
  1:00470h 1       LOCAL_BUS           (R and R/W)
  1:00471h 1x1     PAD4
  1:00472h 1       INT_WLAN            (R/W)
  1:00473h 1x1     PAD5
```

```
1:00474h 4       WINDOW_DATA         (R/W)    ;\
1:00478h 4       WINDOW_WRITE_ADDR   (W)      ;
1:0047Ch 4       WINDOW_READ_ADDR    (W)      ;/
1:00480h 1       HOST_CTRL_SPI_CONFIG (R/W)
1:00481h 1       HOST_CTRL_SPI_STATUS (R/W)
1:00482h 1       NON_ASSOC_SLEEP_EN ;hw2/hw4/hw6 (but didn't exist on AR6001)
1:00483h 1       CPU_DBG_SEL                    ;\DBG, hw4/hw6 only
1:00484h 1x4     CPU_DBG[0..3]                  ;/
1:00488h 1       (HOST_)INT_STATUS2_ENABLE (R/W);\
1:00489h 1x7     PAD6                           ; GMBOX related, hw4/hw6 only
1:00490h 1x8     GMBOX_RX_LOOKAHEAD[0..7]       ;
1:00498h 1       GMBOX_RX_LOOKAHEAD_MUX         ;/
1:00499h 1x359   PAD7
1:00600h 1x512   CIS_WINDOW[0..511]   (R/W?!)       ;SDIO 0:01000h..0:011FFh
1:00800h 800h    Mbox0 Alias (bigger 800h bytes alias)
1:01000h 800h    Mbox1 Alias (bigger 800h bytes alias)
1:01800h 800h    Mbox2 Alias (bigger 800h bytes alias)
1:02000h 800h    Mbox3 Alias (bigger 800h bytes alias)
1:02800h 1800h   Extra Mbox0 Alias "for future usage" (1800h bytes)
1:04000h 1C000h  Unspecified
```

DSi Atheros Wifi - SDIO Function 1 I/O - mbox_wlan_host_reg

Differences between hw2 versus hw4/hw6 (hw4 and hw6 are exactly same):

```
added several new "GMBOX" registers (hw4/hw6)
added new CPU_DBG registers (hw4/hw6)
added three new "UART_HCI_FRAMER_xxx" error bits (hw4/hw6)
renamed "DRAGON_INT" (hw2) to "INT" (hw4/hw6)
renamed "SPI_xxx" (hw2) to "HOST_CTRL_SPI_xxx" (hw4/hw6)
```

1:00000h..000FFh - Mbox0 (100h bytes)
1:00100h..001FFh - Mbox1 (100h bytes)
1:00200h..002FFh - Mbox2 (100h bytes)
1:00300h..003FFh - Mbox3 (100h bytes)
1:00800h..00FFFh - Mbox0 Alias (bigger 800h bytes alias)
1:01000h..017FFh - Mbox1 Alias (bigger 800h bytes alias)
1:01800h..01FFFh - Mbox2 Alias (bigger 800h bytes alias)
1:02000h..027FFh - Mbox3 Alias (bigger 800h bytes alias)
1:02800h..03FFFh - Extra Mbox0 Alias "for future usage" (1800h bytes)
The MBOXes are some sort of FIFOs for transferring data blocks to/from wifi controller. Transfer end seems to be indicated by reading/writing the LAST byte (eg. when sending 4 bytes via Mbox0, data would be usually written to increasing addresses at 1:000FCh..000FFh, or 1:00FFCh..00FFFh when using the bigger Mbox alias) (technically, those addresses are just mirrors of each other, so one could as well write all bytes to the same address, or to random addresses at 1:00000h..000FFh and/or 1:00800h..00FFFh in no specific order; the only special case is that the last byte at 1:000FFh/1:00FFFh seems to be triggering something... like maybe throwing an IRQ at remote side or so).
In total, there appear to be eight MBOXes for SDIO: Four TXFIFOs (from SDIO to Xtensa side), plus four RXFIFOs (from Xtensa to SDIO side). The capacity of the FIFOs is unknown; the 800h-byte spaces would suggest 800h bytes, but the firmware uploader seems to use only max 200h bytes for whatever reason. Hardware tests suggest only 80h bytes MBOX1-3 TXFIFOs, and 8Ah or 3CD0h bytes for MBOX0 TXFIFO (8Ah when writing LAST bytes, 3CD0h otherwise); that results may be disturbed by the firmware trying to process incoming data.
Reading from EMPTY FIFO keeps returning the most recently read value (the last byte before the FIFO got empty).
Writes to FULL FIFO... results are unknown... maybe ignored, and/or producing TIMEOUTs?

1:00400h - HOST_INT_STATUS (R)

1:00418h - (HOST_)INT_STATUS_ENABLE (R/W)
Status.read: 0=No IRQ, 1=IRQ
Enable.read/write: 0=Disable, 1=Enable

```
0-3    MBOX_DATA  MBOX0..3 Data pending (RX FIFO not empty)
4      COUNTER    Secondary IRQ from COUNTER_INT_STATUS
5      INT        Copy of internal CPU's interrupt line (aka DRAGON_INT)
6      CPU        Secondary IRQ from CPU_INT_STATUS
7      ERROR      Secondary IRQ from ERROR_INT_STATUS
```

The status register is read-only (to reset the status bits: read/reset the corresponding MBOXes, or acknowledge the corresponding Secondary IRQ sources).

1:00401h - CPU_INT_STATUS (R/W)
1:00419h - CPU_INT_STATUS_ENABLE (R/W)
Status.read: 0=No IRQ, 1=IRQ
Status.write: 0=No change, 1=Acknowledge
Enable.read/write: 0=Disable, 1=Enable

```
0-7    BIT        Interrupt 0..7 from internal CPU
```

These bits are eight general purpose IRQ signals from the internal CPU (the meaning of the bits depends on software/firmware) (see WLAN_INT_HOST).

1:00402h - ERROR_INT_STATUS (R/W)
1:0041Ah - ERROR_(INT_)STATUS_ENABLE (R/W)
Status.read: 0=No IRQ, 1=IRQ
Status.write: 0=No change, 1=Acknowledge (except bit3)
Enable.read/write: 0=Disable, 1=Enable (except bit3)

```
0      TX_OVERFLOW  (host tried to write to a full MBOX)
1      RX_UNDERFLOW (host tried to read from an empty MBOX)
2      WAKEUP (client has entered ON-state)
3      SPI Error Interrupt   ;STATUS only (not STATUS_ENABLE)  (R)
4      hw4/hw6: UART_HCI_FRAMER_UNDERFLOW             ;\
5      hw4/hw6: UART_HCI_FRAMER_OVERFLOW              ; hw4/hw6 only
6      hw4/hw6: UART_HCI_FRAMER_SYNC_ERROR            ;/
7      -
```

Bit3 can be acknowledged or disabled only via SPI-specific registers from SPI host (that is, probably referring to the HOST_CTRL_SPI_CONFIG and HOST_CTRL_SPI_STATUS registers).

1:00403h - COUNTER_INT_STATUS (R)
1:0041Bh - COUNTER_INT_STATUS_ENABLE (R/W)
Status.read: 0=No IRQ, 1=IRQ
Enable.read/write: 0=Disable, 1=Enable

```
0-7    COUNT[0..7] is nonzero
```

The status register is read-only (to reset the status bits: decrease the COUNT values via COUNT_DEC[0..7]) (unknown if directly writing to COUNT[0..7] does also affect the interrupt bits; probably it does, although official specs suggests that only 00h <--> 01h transitions affect IRQ bits).

1:00406h - HOST_INT_STATUS2 - hw4/hw6 only
1:00488h - (HOST_)INT_STATUS2_ENABLE (R/W) - hw4/hw6 only

```
0      hw4/hw6: GMBOX_DATA                            ;\
1      hw4/hw6: GMBOX_TX_OVERFLOW                     ; hw4/hw6 only
2      hw4/hw6: GMBOX_RX_UNDERFLOW                    ;/
3-7    -
```

1:00404h - MBOX_FRAME (R)

```
0-3    MBOX0..3 contains a SOM (start of message) byte in RX FIFO (1=Yes)
4-7    MBOX0..3 contains a EOM (end of message) byte in RX FIFO   (1=Yes)
```

Note: A SOM byte always follows an EOM byte from the previous message.

1:00405h - RX_LOOKAHEAD_VALID (R)
```
0-3    MBOX0..3 contains at least 4 bytes in RX FIFO (1=Yes)
4-7    -
```

1:00407h - GMBOX_RX_AVAIL - hw4/hw6 only
```
0-6    hw4/hw6: BYTE                                  ;-hw4/hw6 only
7      -
```
uh, a "7bit-byte"? ... or maybe "number of bytes"?

1:00408h..0040Bh - RX_LOOKAHEAD0[0..3] (R)
1:0040Ch..0040Fh - RX_LOOKAHEAD1[0..3] (R)
1:00410h..00413h - RX_LOOKAHEAD2[0..3] (R)
1:00414h..00417h - RX_LOOKAHEAD3[0..3] (R)
```
0-7    MBOX RX FIFO Head-3 byte   ;\what is that?
8-15   MBOX RX FIFO Head-2 byte   ; head "minus" N, or maybe
16-23  MBOX RX FIFO Head-1 byte   ;/head "plus index" N?
24-31  MBOX RX FIFO Head byte
```
Allows to preview the first 1..4 byte(s) from MBOX0..3, without removing the data from the FIFO. The first byte is valid when HOST_INT_STATUS indicates FIFO not empty. The first four bytes are valid when RX_LOOKAHEAD_VALID indicates at least 4 bytes in FIFO.

1:00420h..00427h - COUNT[0..7] (R/W)
This are eight 8bit counter registers for communicating with internal CPU (see LOCAL_COUNT and COUNT_INC in internal I/O map).
```
0-7    Credit Counter Value
```

1:00440h..0045Fh - COUNT_DEC[0..7] (R, or Write=any)
This are eight 32bit registers. Reading or writing the LSB of the 32bit values does decrement the correspondig COUNT register by one. The written value is ignored, reading returns the old COUNT value (before decrement). The decrement doesn't occur if the COUNT is already zero. The corresponding IRQ bit is cleared when COUNT becomes zero.
```
0-7    Credit Counter Value
8-31   Zero? (dummy padding for 32bit access)
```

1:00460h..00467h - SCRATCH[0..7] (R/W)
This are eight 8bit general-purpose registers for communicating with internal CPU (see LOCAL_SCRATCH in internal I/O map).
```
0-7    General Purpose Value
```

1:00468h - FIFO_TIMEOUT (R/W)
Timeout (SDIO Wait duration) for cases when reading from empty MBOXes or writing to full MBOXes. On AR6001, timeout is counted in 1ms units (when CORE_CLK=40MHz) or 0.5ms units (when CORE_CLK=80MHz). Timings for AR6002 are probably same/similar?
```
0-7    Timeout (01h..FFh) (00h=Reserved/don't use)
```

1:00469h - FIFO_TIMEOUT_ENABLE (R/W)
```
0      Enable FIFO Timeouts (0=Disable, 1=Enable)
1-7    -
```

1:0046Ah - DISABLE_SLEEP (R/W)
```
0      Prevent Sleep                       (0=Allow Sleep, 1=Prevent Sleep)
1      Prevent Sleep when Host IRQ pending (0=Allow Sleep, 1=Prevent Sleep)
2-7    -
```

1:0046Eh - LOCAL_BUS_ENDIAN (R/W) (AR6001 only, not hw2/hw4/hw6)
```
0      AR6001 only: (0=Little Endian, 1=Big Endian)   ;-not hw2/hw4/hw6
1-7    -
```

1:00470h - LOCAL_BUS (R and R/W)

```
0-1    Current Chip State (0=Shutdown, 1=On, 2=Sleep, 3=Wakeup) (R)
2      AR6001 only: KEEP_AWAKE (R/W)                  ;\
3      AR6001 only: IO_ENABLE  (R/W)                  ; not hw2/hw4/hw6
4      AR6001 only: SOFT_RESET (R/W)                  ;/
5-7    -
```

1:00472h - INT_WLAN (R/W)

```
0-7    "VECTOR" or interrupt 0..7 ?  (0=No change, 1=Set)
```

Sends IRQs to internal CPU (see WLAN_MBOX_INT_STATUS bit0-7).

1:00474h..00477h - WINDOW_DATA

```
0-31   DATA
```

Used to access the internal Xtensa memory space. The actual memory access occurs when writing the WINDOW_xxx_ADDR registers (see below).

1:00478h..0047Bh - WINDOW_WRITE_ADDR (W) (read: crashes hardware?)
1:0047Ch..0047Fh - WINDOW_READ_ADDR (W) (read: crashes hardware?)

```
0-1    Ignored
2-27   ADDR (in 4-byte steps)
28-31  ?
```

Writing an address to these registers causes 32bit data to be transferred to/from WINDOW_DATA register.

```
For a memory write: First write WINDOW_DATA, then write WINDOW_WRITE_ADDR
For a memory read: First write WINDOW_READ_ADDR, then read WINDOW_DATA
```

The memory transfers seem to occur on writing address LSB (ie. one must first write the upper three ADDR bytes, then write the lower ADDR byte; that requires sending two separate SDIO commands, except, for accesses within the same 100h-byte block, one may get away with a single SDIO command for changing the LSB only, and leaving the MSBs unchanged).
Caution: Trying to read the ADDR registers seems to be somehow crashing the SDIO hardware (causing errors when trying to send any further SDIO commands).

1:00480h - HOST_CTRL_SPI_CONFIG (R/W)

This register can be accessed only via SPI interface.

```
0-1    DATA_SIZE (0=8bit, 1=16bit, 2=32bit, 3=Reserved) (addr = always 16bit)
2      TEST_MODE (0=Normal, 1=Loopback/Echo)
3      INTERRUPT_ENABLE (0=Disable, 1=Enable)
4      SPI_RESET (0=Normal Operation, 1=Reset SPI core)
5      AR6001 only, not AR6002? - SPI_CLK_OFFSET (R)
6      -
7      AR6001 only, not AR6002? - ENDIAN (R/W)
```

1:00481h - HOST_CTRL_SPI_STATUS (R/W)

This register is automatically output on SPI bus after completion of SPI data transfers.

```
0      READY    (0=Command Pending, 1=Completed/Ready)               (R)
1      WR_ERR   (0=Okay, 1=Write-Error)  (write: 0=No change, 1=Ack)  (R/ack)
2      RD_ERR   (0=Okay, 1=Read-Error)   (write: 0=No change, 1=Ack)  (R/ack)
3      ADDR_ERR (0=Okay, 1=Addr-Error)   (write: 0=No change?, 1=Ack?)
4      AR6001 only, not AR6002? - IFF_ERR
5      AR6001 only, not AR6002? - DMA_OVER
6-7    -
```

1:00482h - NON_ASSOC_SLEEP_EN ;hw2/hw4/hw6 (but didn't exist on AR6001)

```
0      BIT
1-7    -
```

1:00483h - CPU_DBG_SEL - hw4/hw6 only

```
0-5    BIT                                            ;-hw4/hw6 only
6-7    -
```

1:00484h..00487h - CPU_DBG[0..3] - hw4/hw6 only

```
0-7    DATA                                          ;-hw4/hw6 only
```

1:00490h..00497h - GMBOX_RX_LOOKAHEAD[0..7] - hw4/hw6 only

```
0-7    DATA                                          ;-hw4/hw6 only
```

1:00498h - GMBOX_RX_LOOKAHEAD_MUX - hw4/hw6 only

```
0      SEL                                           ;-hw4/hw6 only
1-7    -
```

1:00600h..007FFh - CIS_WINDOW[0..511] (R/W?!)

```
0-7    DATA
```

DSi Atheros Wifi Misc

Atheros chip References

Related Atheros chips:

```
AR6001  with MIPS CPU, 18x18 pin BGA package    ;not used in DSi/3DS
AR6002  with Xtensa CPU, 13x13 pin BGA package  ;used in early DSi
AR6013  unknown details (built-in MM3218?)      ;used in later DSi
AR6014  unknown details (similar to AR6013?)    ;used in 3DS
```

There are some datasheets & source code:

```
http://www.datasheetspdf.com/PDF/AR6002/705769/6  ;AR6002 datasheet 56 pages
http://www.datasheetspdf.com/PDF/AR6001X/900300/1 ;AR6001 datasheet 148 pages
http://svn.openmoko.org/developers/nbd/ar6k/      ;AR6K source code
```

The overall hardware registers appear to be same for all AR60xx chips (no matter if they contain MIPS or Xtensa CPUs). The AR6002 datasheet doesn't contain ANY details about hardware registers. The AR6001 datasheet describes SOME hardware registers. And, the AR6K source code contains details about MORE undocumented hardware registers (in some cases listing DIFFERENT addresses as in the datasheet).
The AR6013/AR6014 chips are probably custom designs with some extra MM3218 emulation for NDS games (and possible with some unknown extra hardware features).

BPTWL

Apart from SDIO bus, some wifi functions are also controlled by the BPTWL chip (accessed via I2C bus). The wifi related BPTWL signals are:

```
ATH_TX_H             ;\maybe some/all of these do just indicate traffic
WL_RXPE              ; (for blinking the wifi LED, if it is enabled)
WL_TXPE              ;/
/WIFI_RST            ;-Reset or so
```

That signals seem to be used with BPTWL register 30h, which is allowing to disable the Wifi LED, and might also allow to do things like restting the Wifi hardware, and/or switching between NDS and DSi wifi-modes.

DSi Atheros Wifi - Command Summary

BMI Command Summary (Bootloader Messaging Interface)

```
00h   BMI_NO_COMMAND                          Invalid (ignored)
01h   BMI_DONE                                Launch Firmware
02h   BMI_READ_MEMORY                         Read Memory
03h   BMI_WRITE_MEMORY (normal)               Write Memory
03h   BMI_WRITE_MEMORY (with dest=00001234h)  Segmented Write (not in DSi)
04h   BMI_EXECUTE                             Execute
05h   BMI_SET_APP_START                       Set App Start
06h   BMI_READ_SOC_REGISTER                   Read Register
07h   BMI_WRITE_SOC_REGISTER                  Write Register
```

```
  08h   BMI_GET_TARGET_ID aka BMI_GET_TARGET_INFO  Get Version
  09h   BMI_ROMPATCH_INSTALL                      TCAM/BCAM_xxxxx
  0Ah   BMI_ROMPATCH_UNINSTALL                    TCAM/BCAM_Clr_index_and_xxx
  0Bh   BMI_ROMPATCH_ACTIVATE                     TCAM/BCAM_Set_indices   ;\
  0Ch   BMI_ROMPATCH_DEACTIVATE                   TCAM/BCAM_Clr_indices   ;/
  0Dh   BMI_LZ_STREAM_START                       LZ Uncompress Stream Start
  0Eh   BMI_LZ_DATA                               LZ Data Input
  0Fh   BMI_NVRAM_PROCESS ;not implemented in DSi  Invalid (ignored)
  10h..FFFFFFFFh  Unused                          Invalid (ignored)
```

HTC Services/Handshake

After BMI, and before WMI, there is a "HTC_SERVICES" handshake with following values:

```
  RSVD_SERVICE_GROUP     = 0
  WMI_SERVICE_GROUP      = 1
  HTC_TEST_GROUP         = 254
  HTC_SERVICE_GROUP_LAST = 255
```

MAKE_SERVICE_ID(group,index) = group*100h + index

NOTE: service ID of 0000h is reserved and should never be used

```
  HTC_CTRL_RSVD_SVC = MAKE_SERVICE_ID(RSVD_SERVICE_GROUP,1)
  WMI_CONTROL_SVC   = MAKE_SERVICE_ID(WMI_SERVICE_GROUP,0)  ;control
  WMI_DATA_BE_SVC   = MAKE_SERVICE_ID(WMI_SERVICE_GROUP,1)  ;best effort
  WMI_DATA_BK_SVC   = MAKE_SERVICE_ID(WMI_SERVICE_GROUP,2)  ;background
  WMI_DATA_VI_SVC   = MAKE_SERVICE_ID(WMI_SERVICE_GROUP,3)  ;video
  WMI_DATA_VO_SVC   = MAKE_SERVICE_ID(WMI_SERVICE_GROUP,4)  ;voice
  WMI_MAX_SERVICES  = 5
```

raw stream service (i.e. flash, tcmd, calibration apps):

```
  HTC_RAW_STREAMS_SVC = MAKE_SERVICE_ID(HTC_TEST_GROUP,0)
```

WMI Command Summary (Wireless Module Interface)

```
  0001h  WMI_CONNECT_CMD
  0002h  WMI_RECONNECT_CMD
  0003h  WMI_DISCONNECT_CMD
  0004h  WMI_SYNCHRONIZE_CMD
  0005h  WMI_CREATE_PSTREAM_CMD      ;aka WMI_CRE_PRIORITY_STREAM
  0006h  WMI_DELETE_PSTREAM_CMD      ;aka WMI_DEL_PRIORITY_STREAM
  0007h  WMI_START_SCAN_CMD
  0008h  WMI_SET_SCAN_PARAMS_CMD
  0009h  WMI_SET_BSS_FILTER_CMD      ;aka WMI_BSS_FILTER_CMD
  000Ah  WMI_SET_PROBED_SSID_CMD
  000Bh  WMI_SET_LISTEN_INT_CMD
  000Ch  WMI_SET_BMISS_TIME_CMD
  000Dh  WMI_SET_DISC_TIMEOUT_CMD    ;aka WMI_SET_DISCONNECT_TIMEOUT
  000Eh  WMI_GET_CHANNEL_LIST_CMD    ;reply 000Eh  ;aka WMI_CHANNEL_LIST
  000Fh  WMI_SET_BEACON_INT_CMD
  0010h  WMI_GET_STATISTICS_CMD      ;reply WMI_REPORT_STATISTICS
  0011h  WMI_SET_CHANNEL_PARAMS_CMD  ;aka WMI_CHANNEL_PARAMS_CMD
  0012h  WMI_SET_POWER_MODE_CMD      ;aka WMI_POWER_MODE_CMD
  0013h  WMI_SET_IBSS_PM_CAPS_CMD    ;aka WMI_IBSS_PM_CAPS_CMD
  0014h  WMI_SET_POWER_PARAMS_CMD    ;aka WMI_POWER_PARAMS_CMD
  0015h  WMI_SET_POWERSAVE_TIMERS_POLICY_CMD  ;aka WMI_POWERSAVE...
  0016h  WMI_ADD_CIPHER_KEY_CMD
  0017h  WMI_DELETE_CIPHER_KEY_CMD   ;\
  0018h  WMI_ADD_KRK_CMD             ; ignored dummy commands on DSi
  0019h  WMI_DELETE_KRK_CMD          ;/
  001Ah  WMI_SET_PMKID_CMD
  001Bh  WMI_SET_TX_PWR_CMD
  001Ch  WMI_GET_TX_PWR_CMD          ;aka WMI_TX_PWR ;reply 001Ch
  001Dh  WMI_SET_ASSOC_INFO_CMD
  001Eh  WMI_ADD_BAD_AP_CMD
  001Fh  WMI_DELETE_BAD_AP_CMD
  0020h  WMI_SET_TKIP_COUNTERMEASURES_CMD
  0021h  WMI_RSSI_THRESHOLD_PARAMS_CMD
  0022h  WMI_TARGET_ERROR_REPORT_BITMASK_CMD
```

```
  0023h  WMI_SET_ACCESS_PARAMS_CMD
  0024h  WMI_SET_RETRY_LIMITS_CMD
  0025h  WMI_SET_OPT_MODE_CMD
  0026h  WMI_OPT_TX_FRAME_CMD
  0027h  WMI_SET_VOICE_PKT_SIZE_CMD
  0028h  WMI_SET_MAX_SP_LEN_CMD
  0029h  WMI_SET_ROAM_CTRL_CMD
  002Ah  WMI_GET_ROAM_TBL_CMD ;aka REPORT_ROAM_TBL,TARGET_ROAM_TBL ;reply 100Fh
  002Bh  WMI_GET_ROAM_DATA_CMD       ;reply 1015h ?    ;\
  002Ch  WMI_ENABLE_RM_CMD                             ; not implemented in DSi
  002Dh  WMI_SET_MAX_OFFHOME_DURATION_CMD              ;/
  002Eh  WMI_EXTENSION_CMD   ;prefix for WMIX "Non-wireless extensions"...
  002Eh:2001h  WMIX_DSETOPEN_REPLY_CMD ;reply to 3001h ;\not implemented in DSi
  002Eh:2002h  WMIX_DSETDATA_REPLY_CMD ;reply to 3003h ;/
  002Eh:2003h  WMIX_GPIO_OUTPUT_SET_CMD   ;reply=3006h          ;\
  002Eh:2004h  WMIX_GPIO_INPUT_GET_CMD    ;reply=3005h          ;
  002Eh:2005h  WMIX_GPIO_REGISTER_SET_CMD ;reply=3006h, too     ; GPIO
  002Eh:2006h  WMIX_GPIO_REGISTER_GET_CMD ;reply=3005h, too     ;
  002Eh:2007h  WMIX_GPIO_INTR_ACK_CMD ;reply to 3004h           ;/
  002Eh:2008h  WMIX_HB_CHALLENGE_RESP_CMD ;reply=3007h          ;-HB=heartbeat
  002Eh:2009h  WMIX_DBGLOG_CFG_MODULE_CMD
  002Eh:200Ah  WMIX_PROF_CFG_CMD                       ;\
  002Eh:200Bh  WMIX_PROF_ADDR_SET_CMD                  ;
  002Eh:200Ch  WMIX_PROF_START_CMD                     ; not implemented in DSi
  002Eh:200Dh  WMIX_PROF_STOP_CMD                      ;
  002Eh:200Eh  WMIX_PROF_COUNT_GET_CMD ;reply 3009h    ;/
  002Fh  WMI_SNR_THRESHOLD_PARAMS_CMD
  0030h  WMI_LQ_THRESHOLD_PARAMS_CMD
  0031h  WMI_SET_LPREAMBLE_CMD
  0032h  WMI_SET_RTS_CMD
  0033h  WMI_CLR_RSSI_SNR_CMD
  0034h  WMI_SET_FIXRATES_CMD   ;aka WMI_FIX_RATES_CMD
  0035h  WMI_GET_FIXRATES_CMD        ;reply 0035h
  0036h  WMI_SET_AUTH_MODE_CMD  ;aka WMI_SET_RECONNECT_AUTH_MODE_CMD
;below not in AR6001
  0037h  WMI_SET_REASSOC_MODE_CMD
  0038h  WMI_SET_WMM_CMD
  0039h  WMI_SET_WMM_TXOP_CMD
 ;NOT!   WMI_SET_QOS_SUPP_CMD     ;<-- this NOT here!
  003Ah  WMI_TEST_CMD                                  ;-not implemented in DSi
  003Bh  WMI_SET_BT_STATUS_CMD    ;\AR6002 Bluetooth Coexistence only?
  003Ch  WMI_SET_BT_PARAMS_CMD    ;/
  003Dh  WMI_SET_KEEPALIVE_CMD
  003Eh  WMI_GET_KEEPALIVE_CMD       ;reply 003Eh
  003Fh  WMI_SET_APPIE_CMD  ;aka SET_APP_IE
  0040h  WMI_GET_APPIE_CMD  ;aka GET_APP_IE ;reply=?   ;-not implemented in DSi
  0041h  WMI_SET_WSC_STATUS_CMD   ;aka WSC_REG
  0042h  WMI_SET_HOST_SLEEP_MODE_CMD                   ;\
  0043h  WMI_SET_WOW_MODE_CMD                          ;
  0044h  WMI_GET_WOW_LIST_CMD        ;reply=1018h      ; Wake on Wireless (WOW)
  0045h  WMI_ADD_WOW_PATTERN_CMD                       ;
  0046h  WMI_DEL_WOW_PATTERN_CMD                       ;/
 ;below four as of "AR6kSDK.build_sw.18/include/wmi.h" (from 2006)
 ;0047h  WMI_SET_MAC_ADDRESS_CMD (later moved to F003h)
 ;0048h  WMI_SET_AKMP_PARAMS_CMD (later moved to F004h)
 ;0049h  WMI_SET_PMKID_LIST_CMD  (later moved to F005h)
 ;004Ah  WMI_GET_PMKID_LIST_CMD  (later moved to F006h)
 ;below two are almost always supported (but DSi is somewhat different)
 ;0047h  WMI_SET_FRAMERATES_CMD    ;aka WMI_FRAME_RATES_CMD  ;-cmd 48h on DSi!
 ;0048h  WMI_SET_AP_PS_CMD         ;aka WMI_AP_PS_CMD        ;-not on DSi
Special DSi commands:
  0047h  WMI_START_WHATEVER_TIMER_CMD                  ;-whatever on DSi!
  0048h  WMI_SET_FRAMERATES_CMD    ;aka WMI_FRAME_RATES_CMD  ;-cmd 48h on DSi!
  0049h  WMI_HOST_EXIT_NOTIFY_CMD  ;special DSi/3DS command
```

Special 3DS commands (for Type4 firmware):

```
  004Ah  WMI_AP_HIDDEN_SSID_CMD                          ;01h bytes (1)
  004Bh  WMI_AP_SET_NUM_STA_CMD aka WMI_AP_NUM_STA_CMD   ;01h bytes (1)
  004Ch  WMI_AP_ACL_POLICY_CMD                           ;01h bytes (1)
  004Dh  WMI_AP_ACL_MAC_LIST_CMD aka WMI_AP_ACL_MAC_CMD  ;09h bytes (1+1+6+1)
  004Eh  WMI_AP_CONFIG_COMMIT_CMD                        ;34h bytes (..)
  004Fh  WMI_AP_SET_MLME_CMD                             ;09h bytes (6+2+1)
  0050h  WMI_AP_SET_PVB_CMD     ;ignored on 3DS          ;06h bytes (ignored)
  0051h  WMI_AP_CONN_INACT_CMD                           ;04h bytes (4)
  0052h  WMI_AP_PROT_SCAN_TIME_CMD                       ;08h bytes (4+4)
  0053h  WMI_?                                        ;01h bytes (1)
  0054h  N/A
  0055h  WMI_?                                        ;04h bytes (2+2)
  0056h  WMI_ignored_?                                ;01h bytes (ignored)
  0057h  WMI_?  ;with EVENT_1022h                     ;00h bytes
  0058h  WMI_?     <-- with optional params?          ;var       (2+2+1+1+N)
  0059h  WMI_?       ;only lower 8bit used            ;04h bytes (4; only 1 used)
  005Ah  WMI_?                                        ;01h bytes (1)
  005Bh  WMI_?                                        ;04h bytes (2+1+1)
  005Ch  WMI_?                                        ;04h bytes (1+3)
  005Dh  WMI_?                                        ;0Ah bytes (1+1+2+1+1+2+2)
  005Eh  WMI_?                                        ;01h bytes (1)
  005Fh  WMI_?                                        ;04h bytes (4)
```

Special 3DS commands (for Type5 firmware):

```
  0060h  WMI_?                              ;3Ah bytes
  0061h  WMI_?                              ;28h bytes
  0062h  WMI_?                              ;12h bytes
  0063h  WMI_?                              ;02h bytes
  0064h  WMI_?                              ;03h bytes
  0065h  WMI_?                              ;07h bytes
  0066h  WMI_?                              ;07h bytes
  0067h  WMI_?                              ;03h bytes
  0068h  WMI_?  ;can be with EVENT_1023h    ;01h bytes
  0069h  WMI_?                              ;00h bytes
  006Ah  WMI_?                              ;00h bytes
  006Bh  WMI_?                              ;01h bytes
  006Ch  WMI_?                              ;06h bytes
  006Dh  WMI_?  ;with EVENT_1026h           ;00h bytes
  006Eh  WMI_?                              ;01h bytes
  006Fh  WMI_?                              ;06h bytes
  0070h  WMI_?                              ;01h bytes
  0071h  WMI_?                              ;08h bytes
  0072h  WMI_?                              ;02h bytes
  0073h  WMI_?  ;with EVENT_1027h           ;01h bytes
```

Newer commands

```
  0049h  WMI_SET_QOS_SUPP_CMD  ;<-- this shall be HERE ;\
  004Ah  WMI_SET_IE_CMD  ;new cmd from 2012            ; not implemented in DSi
  08xxh  WILOCITY types          ;\wil6210 stuff       ;
  09xxh  Performance monitoring ;/                     ;/
  8000h  WMI_THIN_RESERVED_START                       ;\
  8000h  WMI_THIN_CONFIG_CMD                           ;
  8001h  WMI_THIN_SET_MIB_CMD                          ; not implemented in DSi
  8002h  WMI_THIN_GET_MIB_CMD ;reply=8001h             ; (thin commands
  8003h  WMI_THIN_JOIN_CMD      ;\newer                ; from wmi_thin.h)
  8004h  WMI_THIN_CONNECT_CMD   ; versions             ;
  8005h  WMI_THIN_RESET_CMD     ;/only                 ;
  8FFFh  WMI_THIN_RESERVED_END                         ;/
```

Developer commands

```
  F000h  WMI_SET_BITRATE_CMD    ;aka WMI_BIT_RATE_CMD
  F001h  WMI_GET_BITRATE_CMD                        ;reply=F001h
  F002h  WMI_SET_WHALPARAM_CMD  ;aka WHAL_PARAMCMD
  F003h  WMI_SET_MAC_ADDRESS_CMD   ;formerly 0047h     ;-not implemented in DSi
  F004h  WMI_SET_AKMP_PARAMS_CMD   ;formerly 0048h
```

```
  F005h  WMI_SET_PMKID_LIST_CMD    ;formerly 0049h
  F006h  WMI_GET_PMKID_LIST_CMD    ;formerly 004Ah   ;reply 1019h
```
Below stuff (F007h..F05Eh) is not implemented in DSi...
```
  F007h  WMI_ABORT_SCAN_CMD
  F008h  WMI_SET_TARGET_EVENT_REPORT_CMD
  F009h  WMI_UNUSED1 or WMI_PYXIS_CONFIG_CMD    ;\Unused (or Pyxis specific
  F00Ah  WMI_UNUSED2 or WMI_PYXIS_OPERATION_CMD ;/commands)
  F00Bh  WMI_AP_HIDDEN_SSID_CMD                  ;\
  F00Ch  WMI_AP_SET_NUM_STA_CMD aka WMI_AP_NUM_STA_CMD   ; AP mode commands
  F00Dh  WMI_AP_ACL_POLICY_CMD                   ;
  F00Eh  WMI_AP_ACL_MAC_LIST_CMD aka WMI_AP_ACL_MAC_CMD  ; (not on DSi, but
  F00Fh  WMI_AP_CONFIG_COMMIT_CMD                ; 3DS type4 supports
  F010h  WMI_AP_SET_MLME_CMD                     ; F00Ah..F013h,
  F011h  WMI_AP_SET_PVB_CMD                      ; renumbered to
  F012h  WMI_AP_CONN_INACT_CMD                   ; 004Ah..0052h)
  F013h  WMI_AP_PROT_SCAN_TIME_CMD               ;
  F014h  WMI_AP_SET_COUNTRY_CMD ;aka WMI_SET_COUNTRY_CMD ;
  F015h  WMI_AP_SET_DTIM_CMD                     ;
  F016h  WMI_AP_MODE_STAT_CMD      ;formerly N/A         ;/
  F017h  WMI_SET_IP_CMD            ;formerly F016h                     ;\
  F018h  WMI_SET_PARAMS_CMD        ;formerly F017h   ;reply=101Fh      ;
  F019h  WMI_SET_MCAST_FILTER_CMD  ;formerly F018h                     ;
  F01Ah  WMI_DEL_MCAST_FILTER_CMD  ;formerly F019h                     ;/
  F01Bh  WMI_ALLOW_AGGR_CMD                                            ;\
  F01Ch  WMI_ADDBA_REQ_CMD                                             ;
  F01Dh  WMI_DELBA_REQ_CMD                                             ;
  F01Eh  WMI_SET_HT_CAP_CMD                                            ;
  F01Fh  WMI_SET_HT_OP_CMD                                             ;
  F020h  WMI_SET_TX_SELECT_RATES_CMD                                   ;
  F021h  WMI_SET_TX_SGI_PARAM_CMD                                      ;
  F022h  WMI_SET_RATE_POLICY_CMD                                       ;/
  F023h  WMI_HCI_CMD_CMD aka WMI_HCI_CMD                               ;\
  F024h  WMI_RX_FRAME_FORMAT_CMD                                       ;
  F025h  WMI_SET_THIN_MODE_CMD                                         ;
  F026h  WMI_SET_BT_WLAN_CONN_PRECEDENCE_CMD                           ;/
  F027h  WMI_AP_SET_11BG_RATESET_CMD                                   ;\
  F028h  WMI_SET_PMK_CMD                                               ;
  F029h  WMI_MCAST_FILTER_CMD                                          ;/
  F02Ah  WMI_SET_BTCOEX_FE_ANT_CMD                    ;\
  F02Bh  WMI_SET_BTCOEX_COLOCATED_BT_DEV_CMD          ;
  F02Ch  WMI_SET_BTCOEX_SCO_CONFIG_CMD                ; AR6003
  F02Dh  WMI_SET_BTCOEX_A2DP_CONFIG_CMD               ; Bluetooth Coexistence
  F02Eh  WMI_SET_BTCOEX_ACLCOEX_CONFIG_CMD            ;
  F02Fh  WMI_SET_BTCOEX_BTINQUIRY_PAGE_CONFIG_CMD     ;
  F030h  WMI_SET_BTCOEX_DEBUG_CMD                     ;
  F031h  WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD       ;
  F032h  WMI_GET_BTCOEX_STATS_CMD  ;reply=1026h..1028h  ;
  F033h  WMI_GET_BTCOEX_CONFIG_CMD ;reply=1027h..1029h  ;/
  F034h  WMI_SET_DFS_ENABLE_CMD        ;aka WMI_SET_DFS_CMD maybe ?   ;\
  F035h  WMI_SET_DFS_MINRSSITHRESH_CMD ;aka WMI_SET_DFS_CMD too ??    ; DFS
  F036h  WMI_SET_DFS_MAXPULSEDUR_CMD   ;aka WMI_SET_DFS_CMD too ??    ;
  F037h  WMI_DFS_RADAR_DETECTED_CMD    ;aka WMI_RADAR_DETECTED_CMD    ;/
  F038h  WMI_P2P_SET_CONFIG_CMD                ;\  ;<-- confirmed to be F038h
  F039h  WMI_WPS_SET_CONFIG_CMD   ;P2P related  ;
  F03Ah  WMI_SET_REQ_DEV_ATTR_CMD ;P2P related  ; P2P CMDS
  F03Bh  WMI_P2P_FIND_CMD                      ;
  F03Ch  WMI_P2P_STOP_FIND_CMD                 ;
  F03Dh  WMI_P2P_GO_NEG_START_CMD              ;
  F03Eh  WMI_P2P_LISTEN_CMD                    ;/
  F03Fh  WMI_CONFIG_TX_MAC_RULES_CMD           ;\  ;<-- claimed to be F040h ?
  F040h  WMI_SET_PROMISCUOUS_MODE_CMD          ;
  F041h  WMI_RX_FRAME_FILTER_CMD               ;
  F042h  WMI_SET_CHANNEL_CMD                   ;/
  F043h  WMI_WAC_ENABLE_CMD aka WMI_ENABLE_WAC_CMD      ;\
  F044h  WMI_WAC_SCAN_REPLY_CMD                         ; WAC commands
```

```
  F045h  WMI_WAC_CTRL_REQ_CMD                          ;/
  F046h  WMI_SET_DIV_PARAMS_CMD aka WMI_DIV_PARAMS_CMD
  F047h  WMI_GET_PMK_CMD         ;reply?               ;\
  F048h  WMI_SET_PASSPHRASE_CMD                        ;/
  F049h  WMI_SEND_ASSOC_RES_CMD      ;aka WMI_SEND_ASSOCRES_CMD     ;\ASSOC
  F04Ah  WMI_SET_ASSOC_REQ_RELAY_CMD ;aka WMI_SET_ASSOCREQ_RELAY    ;/
```

Below uses entirely different numbering in code from 2010 vs 2012...

```
  F04Bh or F04Dh WMI_ACS_CTRL_CMD ;aka WMI_ACS_CTRL_MSG  ;-ACS sub-commands
  F04Ch or F052h WMI_SET_EXCESS_TX_RETRY_THRES_CMD
  F04Dh or N/A   WMI_SET_TBD_TIME_CMD   ;-added for wmiconfig command for TBD
  F04Eh or N/A   WMI_PKTLOG_ENABLE_CMD               ;\Pktlog cmds
  F04Fh or N/A   WMI_PKTLOG_DISABLE_CMD              ;/(code from 2012 only)
  F050h or F053h WMI_P2P_GO_NEG_REQ_RSP_CMD          ;\
  F051h or F054h WMI_P2P_GRP_INIT_CMD                ;
  F052h or F055h WMI_P2P_GRP_FORMATION_DONE_CMD      ;
  F053h or F056h WMI_P2P_INVITE_CMD                  ; More P2P commands
  F054h or F057h WMI_P2P_INVITE_REQ_RSP_CMD          ;
  F055h or F058h WMI_P2P_PROV_DISC_REQ_CMD           ;
  F056h or F059h WMI_P2P_SET_CMD                     ;/
  F057h or F04Bh WMI_GET_RFKILL_MODE_CMD                        ;\RFKILL
  F058h or F04Ch WMI_SET_RFKILL_MODE_CMD ;aka WMI_RFKILL_MODE_CMD ;/
  F059h or F05Ah WMI_AP_SET_APSD_CMD                 ;\More AP commands
  F05Ah or F05Bh WMI_AP_APSD_BUFFERED_TRAFFIC_CMD    ;/
  F05Bh or F05Ch WMI_P2P_SDPD_TX_CMD                 ;\
  F05Ch or F05Dh WMI_P2P_STOP_SDPD_CMD               ; More P2P commands
  F05Dh or F05Eh WMI_P2P_CANCEL_CMD                  ;/
  F05Eh or F04Eh WMI_STORERECALL_CONFIGURE_CMD       ;\Ultra low power
  F05Fh or F04Fh WMI_STORERECALL_RECALL_CMD          ; store/recall commands
  F060h or F050h WMI_STORERECALL_HOST_READY_CMD      ;/
  F061h or F051h WMI_FORCE_TARGET_ASSERT_CMD         ;-
  F062h or N/A   WMI_SET_PROBED_SSID_EX_CMD          ;\
  F063h or N/A   WMI_SET_NETWORK_LIST_OFFLOAD_CMD    ;
  F064h or N/A   WMI_SET_ARP_NS_OFFLOAD_CMD          ;
  F065h or N/A   WMI_ADD_WOW_EXT_PATTERN_CMD         ; NEW stuff
  F066h or N/A   WMI_GTK_OFFLOAD_OP_CMD              ; (code from 2012 only)
  F067h or N/A   WMI_REMAIN_ON_CHNL_CMD              ;
  F068h or N/A   WMI_CANCEL_REMAIN_ON_CHNL_CMD       ;
  F069h or N/A   WMI_SEND_ACTION_CMD                 ;
  F06Ah or N/A   WMI_PROBE_REQ_REPORT_CMD            ;
  F06Bh or N/A   WMI_DISABLE_11B_RATES_CMD           ;
  F06Ch or N/A   WMI_SEND_PROBE_RESPONSE_CMD         ;
  F06Dh or N/A   WMI_GET_P2P_INFO_CMD                ;
  F06Eh or N/A   WMI_AP_JOIN_BSS_CMD                 ;/
  ?              WMI_SET_ADHOC_BSSID_CMD             ;-old, not implemented?
  F0AFh          WMI_AP_PSBUFF_OFFLOAD  ;-QCA4002 chipset on GT202 WiFi module
```

DSi Atheros Wifi - Response Summary

Unknown yet HOW events are transferred; the DSi does send some events via MBOX0, apparently with whatever header(s), which MIGHT be defined in "htc.h". The events spotted on DSi are formatted as so:

```
  00h 2        Values 01h,02h      ;or 00h,02h in case of "obscure values"
  02h 2        Length (excluding trailing zerofilled area)
  04h 2        Values 08h,00h      ;or 0Ch,00h
  06h 2        Event ID (100xh)    ;or "obscure values" (0001h, 0003h, 0201h)
  08h LEN-0Ah  Event/Response Data (Length-0Ah bytes)
  ..  2        Values 02h,06h
  ..  80h-LEN  Zerofilled (till 80h-byte size)  ;sometimes contains a D0h-byte?
```

Some 'events' may occur due to actual hardware events. Other 'events' may occur as response to certain commands - other commands are THEORETICALLY not having any responses, though MAYBE they do still send some sort of confirmation/error values, which MIGHT explain the above "obscure values".

WMI Events/Responses

Events/Responses with same ID as corresponding command

```
  000Eh  WMI_GET_CHANNEL_LIST_CMD
  001Ch  WMI_GET_TX_PWR_CMD
  0035h  WMI_GET_FIXRATES_CMD
  003Eh  WMI_GET_KEEPALIVE_CMD
  F001h  WMI_GET_BITRATE_CMD
```

Events/Responses that could/should exist (but aren't documented)

```
  ?      WMI_GET_APPIE_CMD  ;aka GET_APP_IE            ;\not implemented in DSi
  ?      WMI_AP_MODE_STAT_CMD  ;has reply?             ;/
```

Events/Responses that have REPLY structs defined (but CMDs don't send reply?)

```
  ?      WMI_CRE_PRIORITY_STREAM_REPLY                 ;\
  ?      WMI_DEL_PRIORITY_STREAM_REPLY                 ; not implemented in DSi
  ?      WMI_FRAME_RATES_REPLY                         ;/
```

Events/Responses with special IDs

```
  1001h  WMI_READY_EVENT
  1002h  WMI_CONNECT_EVENT
  1003h  WMI_DISCONNECT_EVENT
  1004h  WMI_BSSINFO_EVENT   ;aka WMI_BSS_INFO
  1005h  WMI_CMDERROR_EVENT  ;aka WMI_CMD_ERROR_EVENT  ;for CMD 01h,11h,16h,26h
  1006h  WMI_REGDOMAIN_EVENT ;aka WMI_REG_DOMAIN_EVENT
  1007h  WMI_PSTREAM_TIMEOUT_EVENT
  1008h  WMI_NEIGHBOR_REPORT_EVENT
  1009h  WMI_TKIP_MICERR_EVENT
  100Ah  WMI_SCAN_COMPLETE_EVENT
  100Bh  WMI_REPORT_STATISTICS_EVENT    ;related to CMD 0010h ?
  100Ch  WMI_RSSI_THRESHOLD_EVENT
  100Dh  WMI_ERROR_REPORT_EVENT ;aka WMI_TARGET_ERROR_REPORT_EVENT
  100Eh  WMI_OPT_RX_FRAME_EVENT ;aka WMI_OPT_RX_INFO
  100Fh  WMI_REPORT_ROAM_TBL_EVENT           ;related to CMD 002Ah ?
  1010h  WMI_EXTENSION_EVENT   ;prefix for WMIX events...
  1010h:3001h  WMIX_DSETOPENREQ_EVENT ;request 2001h   ;\
  1010h:3002h  WMIX_DSETCLOSE_EVENT   ;request close   ; not implemented in DSi
  1010h:3003h  WMIX_DSETDATAREQ_EVENT ;request 2002h   ;/
  1010h:3004h  WMIX_GPIO_INTR_EVENT         ;used (interrupt)
  1010h:3005h  WMIX_GPIO_DATA_EVENT         ;used (reply to 2004h and 2006h)
  1010h:3006h  WMIX_GPIO_ACK_EVENT          ;used (reply to 2003h and 2005h)
  1010h:3007h  WMIX_HB_CHALLENGE_RESP_EVENT ;used (reply to 2008h)
  1010h:3008h  WMIX_DBGLOG_EVENT            ;used (probably related to 2009h)
  1010h:3009h  WMIX_PROF_COUNT_EVENT                    ;-not implemented in DSi
  1011h  WMI_CAC_EVENT
  1012h  WMI_SNR_THRESHOLD_EVENT
  1013h  WMI_LQ_THRESHOLD_EVENT
  1014h  WMI_TX_RETRY_ERR_EVENT
  1015h  WMI_REPORT_ROAM_DATA_EVENT ;related to 002Bh? ;\not implemented in DSi
  1016h  WMI_TEST_EVENT                                 ;/
  1017h  WMI_APLIST_EVENT
  1018h  WMI_GET_WOW_LIST_EVENT              ;reply to CMD 0044h
  1019h  WMI_GET_PMKID_LIST_EVENT            ;reply to CMD F006h
```

Below not in AR6kSDK.build_sw.18, however, "101Ah" is USED on DSi...

```
  101Ah  WMI_CHANNEL_CHANGE_EVENT            ;<-- used on DSi
```

Below some are spotted on 3DS (possibly inofficial stuff?)
(type4=3ds/internet?, macfilter=3ds/localmultiplay?)

```
  101Dh     type4, len=0
  101Fh     type4, len=N*4
  1020h     type4, len=8
  1021h     macfilter, len=10h+?      ;"succes"
  1022h     type4, len=0          ;reply to cmd 0057h ?
  1023h     macfilter, len=4+?    ;reply to cmd 0068h
  1024h     type4, len=0Eh
  1025h     type4, len=1   ;thrown by 3DS type4 variant (works when ignored)
  1026h     macfilter, len=0Ch    ;reply to cmd 006Dh
  1027h     macfilter, len=2+?    ;reply to cmd 0073h
```

Below (101Bh..9004h) not implemented in DSi...

```
  101Bh  WMI_PEER_NODE_EVENT
  101Ch  WMI_PSPOLL_EVENT ;aka WMI_PS_POLL_EVENT  ;AP mode related?
  101Dh  WMI_DTIMEXPIRY_EVENT
  101Eh  WMI_WLAN_VERSION_EVENT
  101Fh  WMI_SET_PARAMS_REPLY_EVENT  ;reply to CMD F018h (reply to "SET" cmd!)
  1020h  WMI_ADDBA_REQ_EVENT
  1021h  WMI_ADDBA_RESP_EVENT
  1022h  WMI_DELBA_REQ_EVENT aka WMI_DELBA_EVENT
  1023h  WMI_TX_COMPLETE_EVENT
  1024h  WMI_HCI_EVENT_EVENT aka WMI_HCI_EVENT
  1025h  WMI_ACL_DATA_EVENT
  1026h  WMI_REPORT_SLEEP_STATE_EVENT   ;formerly N/A
  1027h  WMI_WAPI_REKEY_EVENT           ;formerly N/A, or 1026h if WAPI_ENABLE
  1028h  WMI_REPORT_BTCOEX_STATS_EVENT  ;formerly 1026h/1027h  ;reply to F032h
  1029h  WMI_REPORT_BTCOEX_CONFIG_EVENT ;formerly 1027h/1028h  ;reply to F033h
  102Ah  WMI_GET_PMK_EVENT aka WMI_GET_PMK_REPLY
  102Bh  WMI_DFS_HOST_ATTACH_EVENT              ;\
  102Ch  WMI_DFS_HOST_INIT_EVENT                ;
  102Dh  WMI_DFS_RESET_DELAYLINES_EVENT         ;
  102Eh  WMI_DFS_RESET_RADARQ_EVENT             ;
  102Fh  WMI_DFS_RESET_AR_EVENT                 ; DFS Events
  1030h  WMI_DFS_RESET_ARQ_EVENT                ;
  1031h  WMI_DFS_SET_DUR_MULTIPLIER_EVENT       ;
  1032h  WMI_DFS_SET_BANGRADAR_EVENT            ;
  1033h  WMI_DFS_SET_DEBUGLEVEL_EVENT           ;
  1034h  WMI_DFS_PHYERR_EVENT                   ;/
  1035h  WMI_CCX_RM_STATUS_EVENT                ;-CCX Evants  ;uh, EvAntS?
  1036h  WMI_P2P_GO_NEG_RESULT_EVENT            ;-P2P Events  ;uh, EventS?
  1037h  WMI_WAC_SCAN_DONE_EVENT                ;\
  1038h  WMI_WAC_REPORT_BSS_EVENT               ; WAC
  1039h  WMI_WAC_START_WPS_EVENT                ;
  103Ah  WMI_WAC_CTRL_REQ_REPLY_EVENT           ;/
  103Bh  WMI_RFKILL_STATE_CHANGE_EVENT          ;\RFKILL Events
  103Ch  WMI_RFKILL_GET_MODE_CMD_EVENT          ;/
  103Dh  WMI_P2P_GO_NEG_REQ_EVENT               ;\
  103Eh  WMI_P2P_INVITE_REQ_EVENT               ;
  103Fh  WMI_P2P_INVITE_RCVD_RESULT_EVENT       ;
  1040h  WMI_P2P_INVITE_SENT_RESULT_EVENT       ; More P2P Events
  1041h  WMI_P2P_PROV_DISC_RESP_EVENT           ;
  1042h  WMI_P2P_PROV_DISC_REQ_EVENT            ;
  1043h  WMI_P2P_START_SDPD_EVENT               ;
  1044h  WMI_P2P_SDPD_RX_EVENT                  ;/
  1045h  WMI_SET_HOST_SLEEP_MODE_CMD_PROCESSED_EVENT  ;-avoid AR6003 crash
  8000h  WMI_THIN_EVENTID_RESERVED_START        ;\
  8001h  WMI_THIN_GET_MIB_EVENT                 ; THIN events (wmi_thin.h)
  8002h  WMI_THIN_JOIN_EVENT                    ;
  8FFFh  WMI_THIN_EVENTID_RESERVED_END          ;/
  9000h  WMI_SET_CHANNEL_EVENT                                ;\
  9001h  WMI_ASSOC_REQ_EVENT aka WMI_ASSOCREQ_EVENT           ; More events,
  9002h  WMI_ACS_EVENT    ;generic ACS event                  ; somehow located
  9003h  WMI_REPORT_WMM_PARAMS_EVENT                          ; after THIN area
  9004h  WMI_STORERECALL_STORE_EVENT                          ;/
```

code from 2012 has WMI_WAPI_REKEY_EVENT re-removed again, the two RFKILL Events moved around (placed into the P2P event area), and WMI_REPORT_WMM_PARAMS_EVENT moved to 10xxh, a new WMI_WAC_REJECT_WPS_EVENT at 10xxh, and WMI_SET_HOST_SLEEP_MODE_CMD_PROCESSED_EVENT removed, and WMI_STORERECALL_STORE_EVENT moved to 9003h, and adds more NEW stuff at 9004h-900Dh:

```
  10xxh WMI_REPORT_WMM_PARAMS_EVENT     ;-moved to 10xxh or so
  10xxh WMI_WAC_REJECT_WPS_EVENT        ;-NEW
  9003h WMI_STORERECALL_STORE_EVENT     ;-move to HERE
  9004h WMI_WOW_EXT_WAKE_EVENT          ;\
  9005h WMI_GTK_OFFLOAD_STATUS_EVENT    ;
  9006h WMI_NETWORK_LIST_OFFLOAD_EVENT  ;
```

```
9007h WMI_REMAIN_ON_CHNL_EVENT        ; NEW
9008h WMI_CANCEL_REMAIN_ON_CHNL_EVENT ;
9009h WMI_TX_STATUS_EVENT             ;
900Ah WMI_RX_PROBE_REQ_EVENT          ;
900Bh WMI_P2P_CAPABILITIES_EVENT      ;
900Ch WMI_RX_ACTION_EVENT             ;
900Dh WMI_P2P_INFO_EVENT              ;/
```

DSi Atheros Wifi - Host Interest Area in RAM

Host Interest Area (aka Target Addresses)
Apart from sending BMI and WMI commands via MBOX0, some communication is also done by directly accessing the 100h-byte "Host Interest" area in RAM, this can be done via WINDOW_DATA register (or via BMI commands).
The format of this memory area (defined in "targaddrs.h") is held to be same on all AR60xx chips, however, its BASE ADDRESS may vary on different chips:

```
AR6002_HOST_INTEREST_ADDRESS   = 00500400h  ;older DSi
AR6013_HOST_INTEREST_ADDRESS   = 00520000h  ;newer DSi
AR6014_HOST_INTEREST_ADDRESS   = 00520000h  ;3DS and New3DS
AR6003_HOST_INTEREST_ADDRESS   = 00540600h
MCKINLEY_HOST_INTEREST_ADDRESS = 00400600h
```

That base address can be also found in the DSi's "Wifi Firmware" fileheader.
Aside from the 100h bytes in RAM, there's also another host interest byte in the LOCAL_SCRATCH[0] register; the CHIP_ID register could be also considered as interesting to the host.

Host Interest Entries in RAM ("host_interest_s" structure, 64 words)

```
00h  hi_app_host_interest ;-Pointer to application-defined area, if any.
                          ; (set by Target application during startup)
04h  hi_failure_state ;-Pointer to register dump area after Target crash
08h  hi_dbglog_hdr    ;-Pointer to debug logging header
0Ch  hi_flash_is_present ;Indicates whether or not flash is present on Target
       ;NB: flash_is_present indicator is here not just because it might be
       ;of interest to the Host; but also because it's set early on by
       ;Target's startup asm code and we need it to have a special RAM
       ;address so that it doesn't get reinitialized with the rest of data.
10h  hi_option_flag     ;-Various flags (see below)
14h  hi_serial_enable   ;-Boolean whether to output (additional) TTY messages
18h  hi_dset_list_head  ;-Start address of DataSet index, if any
1Ch  hi_app_start       ;-Override BMI_DONE Target application start address
20h  hi_skip_clock_init          ;\
24h  hi_core_clock_setting       ;
28h  hi_cpu_clock_setting        ; Clock and voltage tuning
2Ch  hi_system_sleep_setting     ;
30h  hi_xtal_control_setting     ;
34h  hi_pll_ctrl_setting_24ghz   ;
38h  hi_pll_ctrl_setting_5ghz    ;
3Ch  hi_ref_voltage_trim_setting ;
40h  hi_clock_info               ;/
44h  hi_bank0_addr_value         ;\Flash configuration overrides, used only
48h  hi_bank0_read_value         ; when firmware is not executing from flash
4Ch  hi_bank0_write_value        ; (when using flash, modify the global
50h  hi_bank0_config_value       ;/variables with equivalent names)
54h  hi_board_data               ;\Pointer to Board Data (eg. from I2C
58h  hi_board_data_initialized   ;/EEPROM) and data present/init flag
5Ch  hi_dset_RAM_index_table     ;-
60h  hi_desired_baud_rate        ;\ ;<-- for TTY/UART (default=9600 decimal)
64h  hi_dbglog_config            ;
68h  hi_end_RAM_reserve_sz       ;
6Ch  hi_mbox_io_block_sz         ;/
70h  hi_num_bpatch_streams       ;-Unused (supposedly was used before 2010)
```

```
  74h  hi_mbox_isr_yield_limit     ;-
  78h  hi_refclk_hz                ;-OSC  ;NN,000,000 decimal (26MHz/40MHz)
 Below seems to be newer stuff... not implemented in DSi... (?)
  7Ch  hi_ext_clk_detected         ;\
  80h  hi_dbg_uart_txpin           ;
  84h  hi_dbg_uart_rxpin           ;
  88h  hi_hci_uart_baud            ;     <-- used in 3DS WMI_READY_EVENT ?
  8Ch  hi_hci_uart_pin_assignments ;/ ;<-- byte[0]=tx, [1]=rx, [2]=rts, [3]=cts
  90h  hi_hci_uart_baud_scale_val  ;\
  94h  hi_hci_uart_baud_step_val   ;/
  98h  hi_allocram_start           ;\
  9Ch  hi_allocram_sz              ;/
  A0h  hi_hci_bridge_flags         ;\
  A4h  hi_hci_uart_support_pins    ;/
         ;NOTE: byte[0]=RESET pin (bit7 is polarity), byte[1..3]=for future use
  A8h  hi_hci_uart_pwr_mgmt_params ;-
         ;Bit[1]:     0=UART FC active low, 1=UART FC active high
         ;Bit[16-31]: wakeup timeout in ms
  ACh  hi_board_ext_data           ;\Pointer to extended board Data, and
  B0h  hi_board_ext_data_config    ;/Config/flags (bit0=valid, bit16-31=size)
  B4h  hi_reset_flag               ;\warmboot flags, valid when [B8h]=12345678h
  B8h  hi_reset_flag_valid         ;/
  BCh  hi_hci_uart_pwr_mgmt_params_ext  ;-bit[0-31]: idle timeout in ms
  C0h  hi_acs_flags                     ;-ACS flags
  C4h  hi_console_flags         ;
  C8h  hi_nvram_state           ;
  CCh  hi_option_flag2          ;
  D0h  hi_sw_version_override   ;\If non-zero, override values sent to Host
  D4h  hi_abi_version_override  ;/in WMI_READY event
  D8h  hi_test_apps_related     ;-test applications flags
  DCh  hi_ota_testscript        ;-location of test script
  E0h  hi_cal_data              ;-location of CAL data
  E4h..FFh      ;reserved
  FCh  reserved, but 3DS sets this to 00000003h along hi_board_data_initialized
```

LOCAL_SCRATCH[0] - AR6K option bits, to enable/disable various features

By default, all option bits are 0.

```
  AR6K_OPTION_BMI_DISABLE   = 01h  ;bit0 Disable BMI comm with Host
  AR6K_OPTION_SERIAL_ENABLE = 02h  ;bit1 Enable UART serial port TTY messages
  AR6K_OPTION_WDT_DISABLE   = 04h  ;bit2 WatchDog Timer override
  AR6K_OPTION_SLEEP_DISABLE = 08h  ;bit3 Disable system sleep
  AR6K_OPTION_STOP_BOOT     = 10h  ;bit4 Stop boot processes (for ATE)
  AR6K_OPTION_ENABLE_NOANI  = 20h  ;bit5 Operate without ANI
  AR6K_OPTION_DSET_DISABLE  = 40h  ;bit6 Ignore DataSets
  AR6K_OPTION_IGNORE_FLASH  = 80h  ;bit7 Ignore flash during bootup
```

targaddr[10h] - hi_option_flag

```
  0     HI_OPTION_TIMER_WAR         ;Enable timer workaround
  1     HI_OPTION_BMI_CRED_LIMIT    ;Limit BMI command credits
  2     HI_OPTION_RELAY_DOT11_HDR   ;Relay Dot11 hdr to/from host
  3     HI_OPTION_MAC_ADDR_METHOD   ;MAC addr method 0=locally administred
                                    ;                1=globally unique addrs
  4     HI_OPTION_ENABLE_RFKILL     ;RF Kill Enable Feature
  5     HI_OPTION_ENABLE_PROFILE    ;Enable CPU profiling
  6     HI_OPTION_DISABLE_DBGLOG    ;Disable debug logging
  7     HI_OPTION_SKIP_ERA_TRACKING ;Skip Era Tracking
  8     HI_OPTION_PAPRD_DISABLE     ;Disable PAPRD (debug)
  9-11  HI_OPTION_NUM_DEV           ;num dev (3bit)
  12-27 HI_OPTION_DEV_MODE          ;dev mode (16bit) (aka 4xMODE, 4xSUBMODE?)
  28    HI_OPTION_NO_LFT_STBL       ;Disable LowFreq LF Timer Stabilization
  29    HI_OPTION_SKIP_REG_SCAN     ;Skip regulatory scan
  30    HI_OPTION_INIT_REG_SCAN     ;Do regulatory scan during init before
                                    ; sending WMI ready event to host
  31    HI_OPTION_FW_BRIDGE         ;Firmware bridging
```

Two bits of hi_option_flag are used to represent 3 (three?) modes

```
HI_OPTION_FW_MODE_IBSS         = 00h   ;IBSS Mode
HI_OPTION_FW_MODE_BSS_STA      = 01h   ;STA Mode
HI_OPTION_FW_MODE_AP           = 02h   ;AP Mode
HI_OPTION_FW_MODE_BT30AMP      = 03h   ;BT30 AMP Mode
```

Two bits of hi_option flag are used to represent 4 submodes (aka DEV_MODE?)

```
HI_OPTION_FW_SUBMODE_NONE      = 00h   ;Normal mode
HI_OPTION_FW_SUBMODE_P2PDEV    = 01h   ;p2p device mode
HI_OPTION_FW_SUBMODE_P2PCLIENT = 02h   ;p2p client mode
HI_OPTION_FW_SUBMODE_P2PGO     = 03h   ;p2p go mode
```

Fw Mode/SubMode Mask

```
.--------.-------.-------.-------.-------.-------.-------.-------.
|  SUB   |  SUB  |  SUB  | SUB   |       |       |       |       |
|MODE[3] |MODE[2]|MODE[1]|MODE[0]|MODE[3]|MODE[2]|MODE[1]|MODE[0]|
|  (2)   |  (2)  |  (2)  |  (2)  |  (2)  |  (2)  |  (2)  |  (2)  |
'--------'-------'-------'-------'-------'-------'-------'-------'
HI_OPTION_FW_MODE_BITS         0x2                                    ;\
HI_OPTION_FW_MODE_MASK         0x3                                    ; MODE
HI_OPTION_FW_MODE_SHIFT        0xC    ;bit12-13 (2bit) per device?    ;
HI_OPTION_ALL_FW_MODE_MASK     0xFF   ;bit12-19 (8bit) per 4 devices? ;/
HI_OPTION_FW_SUBMODE_BITS      0x2                                    ;\
HI_OPTION_FW_SUBMODE_MASK      0x3                                    ; SUB-
HI_OPTION_FW_SUBMODE_SHIFT     0x14   ;bit20-21 (2bit) per device?    ; MODE
HI_OPTION_ALL_FW_SUBMODE_MASK  0xFF00 ;bit20-27 (8bit) per 4 devices? ;
HI_OPTION_ALL_FW_SUBMODE_SHIFT 0x8                                    ;/
```

targaddr[CCh] - hi_option_flag2

```
0     HI_OPTION_OFFLOAD_AMSDU  ;aka OFFLAOD
1     HI_OPTION_DFS_SUPPORT              ;-only in newer code from 2011 or so
2     HI_OPTION_ENABLE_RFKILL            ;\
3     HI_OPTION_RADIO_RETENTION_DISABLE  ;
4     HI_OPTION_EARLY_CFG_DONE           ; only in newer code from 2015 or so
5     HI_OPTION_DISABLE_CDC_MAX_PERF_WAR ;
6     HI_OPTION_USE_EXT_LDO              ;
7     HI_OPTION_DBUART_SUPPORT           ;
8     Reserved?                          ;
9     HT_OPTION_GPIO_WAKEUP_SUPPORT ;HT? ;/
10-31 Reserved
```

targaddr[B4h] - hi_reset_flag (warmboot flags, valid when [B8h]=12345678h)

hi_reset_flag is used to do some stuff when target reset. such as restore app_start after warm reset or preserve host Interest area, or preserve ROM data, literals, etc.

```
0     HI_RESET_FLAG_PRESERVE_APP_START     ;preserve App Start address
1     HI_RESET_FLAG_PRESERVE_HOST_INTEREST ;preserve Host Interest
2     HI_RESET_FLAG_PRESERVE_ROMDATA       ;preserve ROM data
3     HI_RESET_FLAG_PRESERVE_NVRAM_STATE
4     HI_RESET_FLAG_PRESERVE_BOOT_INFO   ;\only in newer code from 2015 or so
5     HI_RESET_FLAG_WARM_RESET           ;/
6-31  Reserved
```

targaddr[C0h] - hi_acs_flags

```
0     HI_ACS_FLAGS_ENABLED           ;ACS is enabled
1     HI_ACS_FLAGS_USE_WWAN          ;Use physical WWAN device
2     HI_ACS_FLAGS_TEST_VAP          ;Use test VAP
3-31  Reserved
```

targaddr[C4h] - hi_console_flags

```
0-2   HI_CONSOLE_FLAGS_UART          ;UART ID     (0=Default)
3     HI_CONSOLE_FLAGS_BAUD_SELECT   ;Baud Select (0=9600, 1=115200)
4-30  Reserved
31    HI_CONSOLE_FLAGS_ENABLE        ;Enable Console
```

targaddr[D8h] - Bitmap for hi_test_apps_related

```
0     HI_TEST_APPS_TESTSCRIPT_LOADED
1     HI_TEST_APPS_CAL_DATA_AVAIL
2-31  Reserved
```

Convert a Target virtual address into a Target physical address

```
AR6002_VTOP(vaddr)   = ((vaddr) & 0x001fffff)  ;\uh, 2Mbyte space?
AR6003_VTOP(vaddr)   = ((vaddr) & 0x001fffff)  ;/(shouldn't that be 4Mbyte?)
MCKINLEY_VTOP(vaddr) = ((vaddr))   ;whatever, maybe uses a different CPU/HW
```

Override REV2 ROM's app start address (whatever crap, doesn't apply to DSi)

```
AR6002_REV2_APP_START_OVERRIDE      0x911A00  ;\
AR6002_REV2_DATASET_PATCH_ADDRESS   0x52D8B0  ; AR6002
AR6002_REV2_APP_LOAD_ADDRESS        0x502070  ;/
AR6003_REV2_APP_START_OVERRIDE      0x944C00  ;\
AR6003_REV2_APP_LOAD_ADDRESS        0x543180  ;
AR6003_REV2_BOARD_EXT_DATA_ADDRESS  0x57E500  ; AR6003 REV2
AR6003_REV2_DATASET_PATCH_ADDRESS   0x57E884  ;
AR6003_REV2_RAM_RESERVE_SIZE        6912      ;/
AR6003_REV3_APP_START_OVERRIDE      0x945D20  ;\
AR6003_REV3_APP_LOAD_ADDRESS        0x545000  ;
AR6003_REV3_BOARD_EXT_DATA_ADDRESS  0x542330  ; AR6003 REV3
AR6003_REV3_DATASET_PATCH_ADDRESS   0x57FEC8  ;
AR6003_REV3_RAM_RESERVE_SIZE        512       ;
AR6003_REV3_RAM_RESERVE_SIZE_TCMD   4352      ;/
```

DSi Atheros Wifi - BMI Bootloader Commands

The BMI commands are used to upload the Wifi Firmware to RAM. Doing that is required because the ROM functions themselves aren't fully functional, and the chip can process only BMI commands or WMI commands (not both at once):

```
BMI Commands --> After RESET
WMI Commands --> After uploading and sending BMI_DONE
```

The AR6002 ROM contains about 40-50% of the program code needed to operate the chip (and most of that ROM code is left unused, until it is getting initialized by the firmware; so the firmware isn't just an "update", it's absolutely required to get the chip working).

DSi Wifi Firmware

On the DSi, the Wifi Firmware is stored in a eMMC file, and it's automatically uploaded by the DSi System Menu (Launcher), so DSi games should be always started with Firmware already installed, and don't need to deal with BMI commands.

DSi SD/MMC Firmware Wifi Firmware

If desired, one could force the chip back to BMI state by issuing a reset, eg. via RESET_CONTROL.Bit8; there might be further ways to issue resets, like SDIO CCCR maybe).

Note: The BMI uploader in DSi is reffering to BMI as "PRE-AUTH" phase, the system menu will be shown even in case of BMI failure, but any such failure will later cause a "WLFIRM 2" error (in sysmenu.log) at time when trying to start a title from within system menu.

BMI Transfer

Before each BMI command, wait for LOCAL_COUNT[4] to become nonzero. Then, write the command and parameters to MBOX0. Then read the response (if any) from MBOX0 (ideally with checking MBOX empty flag before reading response bytes; DSi code doesn't seem to do that though - maybe the SDIO controller is automatically waiting while MBOX empty?).

```
_____________________________ Execute Functions _____________________________
```

BMI_CMD(01h) - BMI_DONE - Launch Firmware

```
  Send 32bit Command (00000001h)
```

Starts the firmware by calling its entrypoint. The default entry is 915000h for AR6002G (or 927000h for AR6013G and AR6014G), alternately a custom entrypoint can be set via BMI_CMD(05h).

BMI_CMD(04h) - BMI_EXECUTE - Execute

```
  Send 32bit Command (00000004h)
  Send 32bit Entrypoint
  Send 32bit Argument
  Receive 32bit Return Value
```

Calls a function on Xtensa CPU. On DSi, this is used to execute the boot stub (for reading the I2C EEPROM data). The main firmware should be started via BMI_DONE, not via BMI_EXECUTE.

BMI_CMD(05h) - BMI_SET_APP_START - Set App Start

```
  Send 32bit Command (00000005h)
  Send 32bit Entrypoint
```

Changes the default entrypoint for BMI_DONE. The DSi doesn't use this feature (and uses the default entrypoint).

```
_____________________________ Read/Write Functions ______________________________
```

BMI_CMD(02h) - BMI_READ_MEMORY - Read Memory

```
  Send 32bit Command (00000002h)
  Send 32bit Address
  Send 32bit Length (should be max 80h or 200h or so (?) due to MBOX size)
  Receive LEN bytes, read from [address and up]
```

Allows to read Xtensa RAM or ROM in byte-units (I/O ports should be read in 32bit units, via BMI_READ_SOC_REGISTER).

BMI_CMD(03h) - BMI_WRITE_MEMORY - Write Memory

```
  Send 32bit Command (00000003h)
  Send 32bit Address (or special value for "Segmented Write", see below)
  Send 32bit Length (should be max 1F4h due to MBOX size)
  Send LEN bytes, written to [address and up]
```

Allows to write Xtensa RAM. Used to upload the stub and data parts of the firmware (the main part of the firmware is uploaded via BMI_LZ_xxx functions).

BMI_CMD(06h) - BMI_READ_SOC_REGISTER - Read Register

```
  Send 32bit Command (00000006h)
  Send 32bit Address
  Receive 32bit Word from [address]
```

Allows to read Xtensa I/O Ports (or RAM or ROM) in 32bit-units. The same effect can be gained via WINDOW_DATA register (which is also working when the Xtensa CPU isn't in BMI bootloader state).

BMI_CMD(07h) - BMI_WRITE_SOC_REGISTER - Write Register

```
  Send 32bit Command (00000007h)
  Send 32bit Address
  Send 32bit Word to [address]
```

Allows to write Xtensa I/O Ports (or RAM) in 32bit-units.

BMI_CMD(0Dh) - BMI_LZ_STREAM_START - LZ Uncompress Stream Start

```
  Send 32bit Command (0000000Dh)
  Send 32bit Destination Start Address for BMI_CMD(0Eh)
```

Sets the destination start address for following BMI_LZ_DATA command(s). Also resets the decompressor to expect the "tag" value in first byte.

BMI_CMD(0Eh) - BMI_LZ_DATA - LZ Data Input

```
  Send 32bit Command (0000000Eh)
  Send 32bit Length (should be max 1F8h due to MBOX size)
  Send LEN compressed bytes, decompressed to incrementing destination address

 _____________________________ ROM Patch Functions _____________________________
```

ROM Patches using the hardware's TCAM/BCAM registers (implemented in hardware, but not actually used by DSi firmware).

BMI_CMD(09h) - BMI_ROMPATCH_INSTALL - TCAM/BCAM_xxxxx

```
  Send 32bit Command (00000009h)
  Send 32bit Target ROM Address
  Send 32bit Target RAM Address or Value (depending on Target Type)
  Send 32bit Size (in bytes)
  Send 32bit Activate (0=Install without activate, 1=Install and activate)
  Receive 32bit PatchID
```

The DSi/3DS do contain TCAM hardware. The TCAM hardware's eight size settings are probably 20h,40h,80h,100h,200h,400h,800h,1000h. For the BCAM hardware, the size must be probably always set to 04h.

BMI_CMD(0Ah) - BMI_ROMPATCH_UNINSTALL - TCAM/BCAM_Clr_index_and_xxx

Uninstall (and deactivate) a previously-installed ROM Patch.

```
  Send 32bit Command (0000000Ah)
  Send 32bit PatchID (to be uninstalled & deactivated)
```

BMI_CMD(0Bh) - BMI_ROMPATCH_ACTIVATE - TCAM/BCAM_Set_indices
BMI_CMD(0Ch) - BMI_ROMPATCH_DEACTIVATE - TCAM/BCAM_Clr_indices

Activate/Deactivate a list of previously-installed ROM Patches.

```
  Send 32bit Command (0000000Bh/0000000Ch)
  Send 32bit Number of patches (N)
  Send Nx32bit List of PatchID's (to be activated/deactivated)
```

```
 _____________________________ Misc Functions _____________________________
```

BMI_CMD(08h) - BMI_GET_TARGET_ID aka BMI_GET_TARGET_INFO - Get Version

```
  Send 32bit Command (00000008h)
  Receive 32bit Value (FFFFFFFFh) ;ROM version (or FFFFFFFFh)
 If above value is FFFFFFFFh then following extra data is appended:
  Receive 32bit Value (0000000Ch) ;total size of extra data
  Receive 32bit Value (20000188h) ;ROM version
  Receive 32bit Value (00000002h) ;TARGET_TYPE (2=AR6002)
```

BMI_CMD(0Fh) - BMI_NVRAM_PROCESS ;not implemented in DSi, Invalid (ignored)

Unknown purpose, said to "process or execute" something in "NVRAM" with a name in "LE format": "Cause Target to search NVRAM (if any) for a segment with the specified name and process it according to NVRAM metadata."

```
  Send 32bit Command (0000000Fh)
  Send 16x8bit Name (16 characters, in "LE format", uh?)
  Receive 32bit Value returned from last executed NVRAM segment (or 0=None)
```

BMI_CMD(00h) - BMI_NO_COMMAND - Invalid (ignored)
BMI_CMD(0Fh..FFFFFFFFh) - N/A - Invalid (ignored)

```
  Send 32bit Command (00000000h, or 0000000Fh..FFFFFFFFh)
```

Segmented Write

The Segmented Write feature is implemented only in newer ROMs (not in DSi with AR6002). It is invoked via BMI_CMD(03h) with destination address set to special value 00001234h.

```
  Send 32bit Command (00000003h) <-- same as Write Memory command
  Send 32bit Address (00001234h) <-- special value for Segmented Write
```

```
  Send 32bit Length (should be max 1F4h due to MBOX size)
  Send LEN bytes, as described below...
```
The transferred data should contain a file header:
```
  00h 4    File ID (544D4753h) ("SGMT")
  04h 4    File Flags (0=Raw, 1=BMI_SGMTFILE_FLAG_COMPRESS)
```
Followed by one or more segments:
```
  00h 4    Destination Address (the actual address, no special value here)
  04h 4    Segment Length (N) (or special value FFFFFFFxh)
  08h N    Data (N bytes)     (no data when N=FFFFFFFxh)
```
Special values for "Segment Length" (all with bit31=1):
```
  FFFFFFFFh ;End of segmented data file (should occur as last segment)
  FFFFFFFEh ;Board Data (write "hi_board_data+Address", instead raw "Address")
  FFFFFFFDh ;Set App Start=Addresss; like BMI_CMD(05h)
  FFFFFFFCh ;Call Address; like BMI_CMD(04h), but without param/return value
```
Compressed Segmented data is said to work in two ways (?):
```
  1) Use BMI_LZ_STREAM_START with Addr=00001234h, followed by BMI_LZ_DATA, or
  2) Use BMI_WRITE_MEMORY with Addr=00001234h and file header Flags=1.
```
Atheros has invented that weird feature for "backwards compatibility & darn convenience", which is both sad and funny because Atheros is definitely unable to <maintain> backwards compatibility (such as keeping the same WMI command/event numbering in firmware revisions), but as it seems, Atheros is believing that "backwards compatibility" is something that can be <invented> by adding obscure features.

DSi Atheros Wifi - MBOX Transfer Headers

MBOX transfers include a small header for distinguishing between Data Packets and Commands/Events, and for indicating the actual length (as opposed to the MBOX transfer length, which is usually garbage-padded to 80h-byte boundary).
The exception are BMI Bootloader commands/responses (which don't have such headers or padding).

SDIO Functions used after BMI init
```
  Func[0:00005h]        R   1     Interrupt Flags (bit1=Function 1 IRQ)
  Func[1:00400h]        R   10h   Interrupt Status, etc. and Lookahead
  Func[1:00800h..00FFF] R/W ..    MBOX0
```

MBOX Send Header/Body
```
  00h 1    Type        (00h=BootInfo, 01h=WMI Command, 02h=Data Packet)
  01h 1    Request Ack (00h=No, 01h=Yes)
  02h 2    Length of entries [06h..end]  (LEN)
  04h 2    ?
  06h LEN  Body (BootInfo/Event/Data)
  ..  ..   Padding to 80h-byte boundary (for SDIO block size 80h)
```
Send Body for BootInfo:
```
  06h ..   related to BE/BK/VI/VO stuff (best effort, background, video, voice)
```
Send Body for WMI Commands:
```
  06h 2  WMI Command Number
  08h .. Parameters
```
Send Body for Data Packets:
```
  06h 2  <Unknown6>?? 0000h or 1C00h    ;maybe whatever ?
  08h 6  Destination (MAC Addr of Router) (or FF:FF:FF:FF:FF:FF=Broadcast)
  0Eh 6  Source      (MAC Addr of DSi console)
  14h 2  Length Data at [16h...]  ;(LEN1-10h)   ;<-- BIG-ENDIAN !!!
  16h .. Data (usually starting with LLC stuff, ie. AAh,AAh,03h,00h...)
```
Send Body for empty Data Packets (used when changing WPA/WPA2 CIPHER keys):
```
  06h 1  Garbage (usually LSB of a previously used WMI command number)
  07h 1  Unknown (02h)
```

MBOX Receive Header/Body
```
  00h      1    Type        (00h=Ack only, 01h=WMI Event, 02h=Data Packet)
```

```
  01h      1    Ack Present (00h=No, 02h=Yes)
  02h      2    Length of entries [06h..end]     (LEN1+LEN2)
  04h      1    Length of Ack at [06h+LEN1..end] (LEN2) ;garbage when [01h]=00h
  05h      1    ? (00h,20h?,7Fh,F4h,FFh)
  06h      LEN1 Body (Event/Data)
  06h+LEN1 LEN2 Ack List
  ..       ..   Padding to 80h-byte boundary (for SDIO block size 80h)
```
Receive Body for Ack only:
```
  06h -  N/A (or boot request info; occurs only shortly after BMI)
```
Receive Body for Events:
```
  06h 2  WMI Event Number
  08h .. Parameters
```
Receive Body for Data Packets:
```
  06h 1  RSSI (Received Signal Strength Indicator) (00h..3Ch) (aka 0..60)
  07h 1  <Unknown7> 00h
  08h 6  Destination (MAC Addr of DSi console) (or FF:FF:FF:FF:FF:FF=Broadcast)
  0Eh 6  Source      (MAC Addr of Router)
  14h 2  Length of Data entries at [16h..(end-xtracrap)]   ;<-- BIG-ENDIAN !!!
  16h .. Data (usually LLC stuff, ie. AAh,AAh,03h.. or, once F0h,F0h,03h..??)
```
Ack List (aka Trailer) (can contain multiple List items):
```
  X+00h  List Item Type (01h=Ack, 02h=Lookahead)
  X+01h  List Item Len (02h*N for Ack's, 06h for Lookahead)
  X+02h  List Item data
```
Ack Items (can contain multiple Ack items):
```
  Y+00h  Ack Item Type (01h=Command Ack, 02h=Data Ack, 05h=Data Ack, too?)
  Y+01h  Ack Item Count? (usually 1) (or 2 for double-ack?)
```
Lookahead Item (if valid: allows to omit reading Func[1:00408h]):
```
  Z+00h  Lookahead valid ID1 (00h=No, 55h=Valid)
  Z+01h  Lookahead MBOX0 (or garbage if ID1/ID2 not valid)
  Z+05h  Lookahead valid ID2 (00h=No, AAh=Valid)
```

Notes

The send/receive DATA headers are apparently containing incomplete wifi "Frame Headers":
```
  without 3rd address field
  without WEP entries (those are apparently automatically inserted)
  without Frame Control, Duration/ID, Sequence Control
  there seem to be only Data Frames (no Managment/Control Frames)
```
That, unless some of the missing info would be encoded in the <Unknown> header entries.

DSi Atheros Wifi - WMI Misc Commands

WMIcmd(0004h) - WMI_SYNCHRONIZE_CMD

Parameters (01h bytes):
```
  00h  A_UINT8  1  dataSyncMap          ;00h, 01h, or 09h used?
```

WMIcmd(0009h) - WMI_SET_BSS_FILTER_CMD ;aka WMI_BSS_FILTER_CMD

Parameters (08h bytes):
```
  00h  A_UINT8  1  bssFilter;                     /* see WMI_BSS_FILTER
  01h  A_UINT8  1  reserved1;                     /* For alignment
  02h  A_UINT16 2  reserved2;                     /* For alignment
  04h  A_UINT32 4  ieMask;
```
WMI_BSS_FILTER values:
```
  NONE_BSS_FILTER        = 00h  ;no beacons forwarded
  ALL_BSS_FILTER         = 01h  ;all beacons forwarded
  PROFILE_FILTER         = 02h  ;only beacons matching profile
  ALL_BUT_PROFILE_FILTER = 03h  ;all but beacons matching profile
  CURRENT_BSS_FILTER     = 04h  ;only beacons matching current BSS
  ALL_BUT_BSS_FILTER     = 05h  ;all but beacons matching BSS
```

```
  PROBED_SSID_FILTER     = 06h  ;beacons matching probed ssid
  LAST_BSS_FILTER        = 07h  ;marker only
```

WMIcmd(000Ah) - WMI_SET_PROBED_SSID_CMD

Parameters (23h bytes):

```
  00h  A_UINT8  1   entryIndex;      /* 0 to MAX_PROBED_SSID_INDEX
  01h  A_UINT8  1   flag;            /* WMI_SSID_FLG
  02h  A_UINT8  1   ssidLength;
  03h  A_UINT8  32  ssid[32];
```

#define MAX_PROBED_SSID_INDEX = 15

WMI_SSID_FLAG values:

```
  DISABLE_SSID_FLAG  = 00h    /* disables entry
  SPECIFIC_SSID_FLAG = 01h    /* probes specified ssid
  ANY_SSID_FLAG      = 02h    /* probes for any ssid
```

WMIcmd(000Bh) - WMI_SET_LISTEN_INT_CMD

Parameters (04h bytes):

```
  00h  A_UINT16 2   listenInterval;
  02h  A_UINT16 2   numBeacons;
```

The Listen interval is between 15 and 3000 TUs

```
  MIN_LISTEN_INTERVAL = 15     ;min = 15
  MAX_LISTEN_INTERVAL = 5000   ;max = 5000 or 3000, uh?
  MIN_LISTEN_BEACONS  = 1
  MAX_LISTEN_BEACONS  = 500
```

WMIcmd(000Ch) - WMI_SET_BMISS_TIME_CMD

Parameters (04h bytes):

```
  00h  A_UINT16 2   bmissTime;
  02h  A_UINT16 2   numBeacons;
```

Valid values are between 1000 and 5000 TUs

```
  MIN_BMISS_TIME    = 1000
  MAX_BMISS_TIME    = 5000
  MIN_BMISS_BEACONS = 1
  MAX_BMISS_BEACONS = 50
```

WMIcmd(000Fh) - WMI_SET_BEACON_INT_CMD

Parameters (02h bytes):

```
  00h  A_UINT16 2    beaconInterval;
```

WMIcmd(001Ah) - WMI_SET_PMKID_CMD

Parameters:

```
  00h  A_UINT8  6   bssid[ATH_MAC_LEN];
  06h  A_UINT8  1   enable;                 /* PMKID_ENABLE_FLG
  07h  A_UINT8  16  pmkid[WMI_PMKID_LEN];
```

#define WMI_PMKID_LEN = 16

PMKID_ENABLE_FLG values:

```
  PMKID_DISABLE = 0
  PMKID_ENABLE  = 1
```

WMIcmd(001Dh) - WMI_SET_ASSOC_INFO_CMD

Parameters:

```
  00h  A_UINT8  1   ieType
  01h  A_UINT8  1   bufferSize
  02h  A_UINT8  N*1 assocInfo[1]         ;up to WMI_MAX_ASSOC_INFO_LEN
```

A maximum of 2 private IEs can be sent in the [Re]Assoc request.

A 3rd one, the CCX version IE can also be set from the host.

```
  WMI_MAX_ASSOC_INFO_TYPE   = 2
  WMI_CCX_VER_IE            = 2   /* ieType to set CCX Version IE
  WMI_MAX_ASSOC_INFO_LEN    = 240
```

WMIcmd(001Eh) - WMI_ADD_BAD_AP_CMD
Parameters (07h bytes):
```
  00h  A_UINT8  1  badApIndex            ;0 to WMI_MAX_BAD_AP_INDEX
  01h  A_UINT8  6  bssid[ATH_MAC_LEN]
 WMI_MAX_BAD_AP_INDEX     = 1
```

WMIcmd(001Fh) - WMI_DELETE_BAD_AP_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1  badApIndex            ;0 to WMI_MAX_BAD_AP_INDEX
```

WMIcmd(0023h) - WMI_SET_ACCESS_PARAMS_CMD
Parameters:
```
  00h  A_UINT16 2  txop           ;in units of 32 usec
  02h  A_UINT8  1  eCWmin
  03h  A_UINT8  1  eCWmax
  04h  A_UINT8  1  aifsn
  05h  A_UINT8  1  ac
 WMI_DEFAULT_TXOP_ACPARAM   = 0       /* implies one MSDU
 WMI_DEFAULT_ECWMIN_ACPARAM = 4       /* corresponds to CWmin of 15
 WMI_DEFAULT_ECWMAX_ACPARAM = 10      /* corresponds to CWmax of 1023
 WMI_MAX_CW_ACPARAM         = 15      /* maximum eCWmin or eCWmax
 WMI_DEFAULT_AIFSN_ACPARAM  = 2
 WMI_MAX_AIFSN_ACPARAM      = 15
```

WMIcmd(0025h) - WMI_SET_OPT_MODE_CMD
Parameters (01h bytes on DSi?, but parameter structure is undocumented):
```
  00h  A_UINT8     optMode (documented in code from 2008 only)
```
OPT_MODE_TYPE values:
```
  SPECIAL_OFF      = unknown (maybe 0 or 1 or so)  ;\
  SPECIAL_ON       = SPECIAL_OFF+1                 ; code from 2008 only
  PYXIS_ADHOC_ON   = SPECIAL_OFF+2                 ; (removed/undoc in 2010)
  PYXIS_ADHOC_OFF  = SPECIAL_OFF+3                 ;/
```

WMIcmd(0026h) - WMI_OPT_TX_FRAME_CMD
Unknown/undocumented purpose. Unknown if/how this is implemented in DSi. The DSi does redirect cmd 0026h to a function handler, but that handler does just trigger a misalign exception. However, the DSi does also have a function for handling the command (with "opt_tx_frame_cmd" string in AR6013/AR6014), but that function doesn't seem to be called upon command number 0026h. Maybe the CMD is triggered upon special flags in the SDIO transfer block header, rather than by the command number?
Parameters (11h bytes on DSi)
```
  Unknown (11h bytes, but parameter structure is undocumented)
```
See also: WMI_OPT_RX_FRAME_EVENT (counterpart in opposite direction), and WMI_SET_OPT_MODE_CMD.

WMIcmd(0027h) - WMI_SET_VOICE_PKT_SIZE_CMD
Parameters (02h bytes):
```
  00h  A_UINT16 2   voicePktSize
```

WMIcmd(0028h) - WMI_SET_MAX_SP_LEN_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1  maxSPLen
```
APSD_SP_LEN_TYPE values:
```
    DELIVER_ALL_PKT = 00h
    DELIVER_2_PKT   = 01h
    DELIVER_4_PKT   = 02h
    DELIVER_6_PKT   = 03h
```

WMIcmd(002Eh) - WMI_EXTENSION_CMD ;prefix for WMIX "Non-wireless extensions"
Parameters:
```
  00h  UNIT32?  4    WMIX Command (values 2001h and up)  ;WMIX_CMD_HDR
  04h  ...      ..   WMIX Parameter(s)
```
Prefix for WMIX commands.

WMIcmd(002Eh:2008h) - WMIX_HB_CHALLENGE_RESP_CMD ;reply=3007h ;HB=heartbeat
```
  00h  A_UINT32 4    cookie    ;usually increasing 1,2,3,4,5,etc.
  04h  A_UINT32 4    source    ;usually 0
```
Heartbeat Challenge Response command, this is said to be an "Error Detection support" feature.
There's also a WMI_SET_HB_CHALLENGE_RESP_PARAMS_CMD structure defined in wmi.h (not in wmix.h):
```
  00h  A_UINT32 frequency    ;\unknown purpose, sounds like command/params,
  04h  A_UINT8  threshold    ;/but there's no WMIcmd(xxxxh) value assigned
```
See also: WMIX_HB_CHALLENGE_RESP_EVENT

WMIcmd(0031h) - WMI_SET_LPREAMBLE_CMD
Parameters (01h bytes on DSi?, but other sources claim 02h bytes):
```
  01h 02h   <---- total size (on DSi it's 01h, ie. left column)
  00h 00h  A_UINT8  1   status
  --  01h  A_UINT8  1   preamblePolicy
```
WMI_LPREAMBLE_STATUS values:
```
  WMI_LPREAMBLE_DISABLED = 0
  WMI_LPREAMBLE_ENABLED  = 1
```
WMI_PREAMBLE_POLICY values:
```
  WMI_IGNORE_BARKER_IN_ERP       = 0
  WMI_DONOT_IGNORE_BARKER_IN_ERP = 1
```

WMIcmd(0032h) - WMI_SET_RTS_CMD
Parameters (02h bytes):
```
  00h  A_UINT16 2   threshold
```

WMIcmd(0036h) - WMI_SET_AUTH_MODE_CMD ;aka
WMI_SET_RECONNECT_AUTH_MODE_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1   mode
```
WMI_AUTH_MODE values:
```
  RECONN_DO_AUTH  = 00h
  RECONN_NOT_AUTH = 01h
```
Set authentication mode

WMIcmd(0037h) - WMI_SET_REASSOC_MODE_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1   mode
```
WMI_REASSOC_MODE values:
```
  REASSOC_DO_DISASSOC    = 00h
  REASSOC_DONOT_DISASSOC = 01h
```
Set authentication(?) mode (uh, so SET_REASSOC is same as SET_AUTH?)

WMIcmd(0038h) - WMI_SET_WMM_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1   status
```
WMI_WMM_STATUS values:
```
  WMI_WMM_DISABLED = 0
  WMI_WMM_ENABLED  = 1
```
WMM stands for Wi-Fi Multimedia (WMM) aka Wireless Multimedia Extensions (WME) aka some QOS related stuff?

"WMM prioritizes traffic according to four Access Categories (AC): voice (AC_VO), video (AC_VI), best effort (AC_BE), and background (AC_BK)."

WMIcmd(0039h) - WMI_SET_WMM_TXOP_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1   txopEnable
```
WMI_TXOP_CFG values:
```
  WMI_TXOP_DISABLED = 0
  WMI_TXOP_ENABLED  = 1
```

WMIcmd(003Dh) - WMI_SET_KEEPALIVE_CMD
Parameters (01h bytes):
```
  00h  A_UINT8  1   keepaliveInterval
```

WMIcmd(003Eh) - WMI_GET_KEEPALIVE_CMD ;reply 003Eh
Parameters (claimed to be as so?):
```
    A_BOOL   4   configured
    A_UINT8  1   keepaliveInterval
```
Uh, the command does probably not have any parameters at all, and above is probably meant to be the Reply structure, not the Parameter structure?

WMIcmd(003Fh) - WMI_SET_APPIE_CMD ;aka WMI_SET_APP_IE_CMD
Can be used to add custom Information Elements (IE) to beacons?
Parameters (old=2+N bytes, or new=4+N bytes):
```
 Older version (used on DSi, and on older 3DS firmwares):
  00h  A_UINT8  1    mgmtFrmType ;one of WMI_MGMT_FRAME_TYPE
  01h  A_UINT8  1    ieLen     ;Length of the IE to be added to the MGMT frame
  02h  A_UINT8  N*1  ieInfo[1] ;
 Newer version (used on newer 3DS firmwares):
  00h  A_UINT8  1    mgmtFrmType ;one of WMI_MGMT_FRAME_TYPE
  01h  A_UINT8  1    reserved (zero, actually USED as MSB of above type value)
  02h  A_UINT16 2    ieLen     ;Length of the IE to be added to the MGMT frame
  04h  A_UINT8  N*1  ieInfo[1]
```
WMI_MGMT_FRAME_TYPE values:
```
  WMI_FRAME_BEACON     = 0   ;\old version (with 8bit ieLen):
  WMI_FRAME_PROBE_REQ  = 1   ;  supports type 0,1,2,3,4 (or 5..FFh=same as 4)
  WMI_FRAME_PROBE_RESP = 2   ; new version (with 16bit ieLen):
  WMI_FRAME_ASSOC_REQ  = 3   ;  supports type 1,3 only (ignores all other)
  WMI_FRAME_ASSOC_RESP = 4   ;/
```
Add Application specified IE to a management frame
```
  WMI_MAX_IE_LEN = 78    ;older versions (with 8bit ieLen) ;<-- actual limit
  WMI_MAX_IE_LEN = 255   ;older versions (with 8bit ieLen) ;<-- incorrect
  WMI_MAX_IE_LEN = 1024  ;newer versions (with 16bit ieLen) (from 2014)
```
Uh, if that newer 3DS firmware is also used in DSi mode (?) then the changed parameter format would be incompatible with DSi software (the APPIE command is rarely (perhaps never) used though).

WMIcmd(0041h) - WMI_SET_WSC_STATUS_CMD ;aka WSC_REG
Parameters:
```
  Unknown (none? or maybe UINT8 or so, maybe with values listed below)
  DSi uses 1 byte parameter.
 WSC_REG_ACTIVE   = 1
 WSC_REG_INACTIVE = 0
```
Notify the WSC registration status to the target.
WSC means Wifi Simple Config?

WMIcmd(0049h) - WMI_HOST_EXIT_NOTIFY_CMD ;special DSi/3DS command
Parameters:
```
  None
```

This is apparently a special nintendo command, supported on DSi and 3DS (supported on AR6002/AR6013/AR6014, whereof, AR6013/AR6014 have it bundled with a "wmi_host_exit_notify_cmd" message, so, this is apparently not the official WMI_SET_QOS_SUPP_CMD from atheros).

WMIcmd(0049h) - WMI_SET_QOS_SUPP_CMD ;not implemented in DSi/3DS...?
Parameters:
```
  00h  A_UINT8  1   status;
```
QOS or QoS stands for "Quality of service"?
Note: Some older source code from 2008 did have this command accidently inserted between WMIcmd(0039h and 003Ah), thereby smashing the numbering for WMIcmd(003Ah..004xh), that issue is fixed in newer source code from 2010.

...Developer commands...

WMIcmd(F002h) - WMI_SET_WHALPARAM_CMD ;aka WHAL_PARAMCMD
Parameters:
```
  00h  A_UINT8  1   whalCmdId;    ;see WHAL_CMDID enumeration
  01h  A_UINT8  ..  data[1];      ;aka SETCABTO structure ?
```
Generic Hal Interface for setting hal parameters.
Add new Set HAL Param cmdIds here for newer params.
WHAL_CMDID values (only one defined):
```
  WHAL_SETCABTO_CMDID = 1
```
WHAL_SETCABTO_PARAM structure:
```
  A_UINT8 cabTimeOut;
```

WMIcmd(F004h, or formerly 0048h) - WMI_SET_AKMP_PARAMS_CMD
Parameters (04h bytes):
```
  00h  A_UINT32 4  akmpInfo;
```
WMI_AKMP_MULTI_PMKID_EN = 000001h

WMIcmd(F005h, or formerly 0049h) - WMI_SET_PMKID_LIST_CMD
Parameters (01h bytes on DSi?, but below would be 4+N*? bytes?):
```
  00h  A_UINT32  4     numPMKID;
  04h  WMI_PMKID N*..  pmkidList[WMI_MAX_PMKID_CACHE];
```
#define WMI_MAX_PMKID_CACHE = 8
WMI_PMKID structure:
```
  A_UINT8  pmkid[WMI_PMKID_LEN];
```

WMIcmd(F006h, or formerly 004Ah) - WMI_GET_PMKID_LIST_CMD ;reply 1019h
Parameters:
```
  Unknown (none?)
```
Reply: WMI_GET_PMKID_LIST_EVENT ;aka WMI_PMKID_LIST_REPLY
```
  00h  A_UINT32  4    numPMKID;
  04h  A_UINT8   N*6  bssidList[ATH_MAC_LEN][1];
  ..   WMI_PMKID N*1  pmkidList[1];
```
"Following the Number of PMKIDs is the list of PMKIDs" uh, what exactly?
WMI_PMKID structure:
```
    A_UINT8 pmkid[WMI_PMKID_LEN];
```

DSi Atheros Wifi - WMI Misc Events

WMIevent(000Eh) - WMI_GET_CHANNEL_LIST_CMD
WMIevent(001Ch) - WMI_GET_TX_PWR_CMD
WMIevent(0035h) - WMI_GET_FIXRATES_CMD
WMIevent(003Eh) - WMI_GET_KEEPALIVE_CMD

WMIevent(F001h) - WMI_GET_BITRATE_CMD
Events/Responses with same ID as corresponding command.
See corresponding commands for description of reply data.

WMIevent(1001h) - WMI_READY_EVENT
This event exists with numerous (incorrect) definitions, claiming the event data to be 07h, 0Bh, 0Ch, 0Fh, or 10h bytes in length:
Event Data (0Ch bytes on DSi, as returned by DSi firmware):
```
  00h  A_UINT8  6  macaddr[ATH_MAC_LEN]   ;MAC addr of DSi console
  06h  A_UINT8  1  phyCapability    (=02h aka "11G")  ;WMI_PHY_CAPABILITY
  07h  A_UINT8  1  unused/padding
  08h  A_UINT32 4  version (=2100007Bh/2300006Ch) (firmware version)
```
Event Data (07h bytes, exists in AR6002 ROM, but overridden by DSi firmware):
```
  00h  A_UINT8  6  macaddr[ATH_MAC_LEN]
  06h  A_UINT8  1  phyCapability                      ;WMI_PHY_CAPABILITY
```
Event Data (0Bh bytes, accidently claimed to be so in source code from 2008):
```
  00h  A_UINT32 4  version
  04h  A_UINT8  6  macaddr[ATH_MAC_LEN]
  0Ah  A_UINT8  1  phyCapability                      ;WMI_PHY_CAPABILITY
```
Event Data (0Fh bytes, accidently claimed to be so in source code from 2010):
```
  00h  A_UINT32 4  sw_version
  04h  A_UINT32 4  abi_version
  08h  A_UINT8  6  macaddr[ATH_MAC_LEN]
  0Eh  A_UINT8  1  phyCapability                      ;WMI_PHY_CAPABILITY
```
Event Data (10h bytes, as done by 3DS function in wifi firmware RAM):
```
  00h  A_UINT8  6  macaddr[ATH_MAC_LEN]   ;MAC addr of DSi console
  06h  A_UINT8  1  phyCapability    (=02h aka "11G")  ;WMI_PHY_CAPABILITY
  07h  A_UINT8  1  unused/padding
  08h  A_UINT32 4  version (230000B3h) (firmware version)
  0Ch  A_UINT16 2  whus         ;=[004134h] ;some I/O register ???
  0Eh  A_UINT16 2  what         ;=[520088h] ;hi_hci_uart_baud  ???
```
WMI_PHY_CAPABILITY values (maybe the "11" is related to "IEEE 802.11"?):
```
  WMI_11A_CAPABILITY   = 1
  WMI_11G_CAPABILITY   = 2
  WMI_11AG_CAPABILITY  = 3
  WMI_11NA_CAPABILITY  = 4
  WMI_11NG_CAPABILITY  = 5
  WMI_11NAG_CAPABILITY = 6
```

WMIevent(1004h) - WMI_BSSINFO_EVENT aka WMI_BSS_INFO
Event Data (0Ch+N bytes or 10h+N bytes on DSi, depending on version setting):
```
 When version<2:
  00h  A_UINT16 2   channel      ;in MHz                ;\
  02h  A_UINT8  1   frameType    ;see WMI_BI_FTYPE      ;
  03h  A_UINT8  1   snr      ;eg. 33h                   ; WMI_BSS_INFO_HDR
  04h  A_INT16  2   rssi     ;eg. FFD4h aka "33h-95"    ; version 1 (10h bytes)
  06h  A_UINT8  6   bssid[ATH_MAC_LEN]                  ;
  0Ch  A_UINT32 4   ieMask                              ;/
  10h  BODY     ... beacon or probe-response frame body ;-Body (timestamp etc)
 When version>=2:
  00h  A_UINT16 2   channel      ;in MHz                ;\
  02h  A_UINT8  1   frameType    ;see WMI_BI_FTYPE      ; WMI_BSS_INFO_HDR2
  03h  A_UINT8  1   snr   (implies "rssi=snr-95" in v2) ; version 2 (0Ch bytes)
  04h  A_UINT8  6   bssid[ATH_MAC_LEN]                  ;
  0Ah  A_UINT16 2   ieMask         (only 2 bytes in v2) ;/
  0Ch  BODY     ... beacon or probe-response frame body ;-Body (timestamp etc)
```
BUG: The "snr" value can be negative, but isn't sign-expanded to 16bit, and the and the rssi value at [04h] does then contain nonsense (eg. spotted bytes at[03h..05h] are "FC 9D 00").
The BODY starts with 64bit Timestamp, 16bit BeaconInterval, etc. For details, see:
DS Wifi IEEE802.11 Managment Frames (Type=0)
The size of the BODY varies per access point (from around 64 bytes to exceeding 256 bytes).

The DSi firmware contains code for both of the above HDR versions (unknown if/when/how it does use which version; the DSi Browser receives the older type, with 10h+N bytes).
BSS Info Event.
Mechanism used to inform host of the presence and characteristic of wireless networks present. Consists of bss info header followed by the beacon or probe-response frame body. The 802.11 header is not included.
BSS INFO HDR version 2.0:
With 6 bytes HTC header and 6 bytes of WMI header WMI_BSS_INFO_HDR cannot be accomodated in the removed 802.11 management header space.

```
  - Reduce the ieMask to 2 bytes as only two bit flags are used
  - Remove rssi and compute it on the host. rssi = snr - 95
```

WMI_BI_FTYPE values:

```
  BEACON_FTYPE      = 01h
  PROBERESP_FTYPE   = 02h
  ACTION_MGMT_FTYPE = 03h
  PROBEREQ_FTYPE    = 04h
```

BSS_ELEMID values (unclear WHAT that is, maybe the "ieMask" stuff, which said to use "two bit flags"):

```
  BSS_ELEMID_CHANSWITCH = 01h    ;value for bit0? (or bit-number for bit1?)
  BSS_ELEMID_ATHEROS    = 02h    ;value for bit1? (or bit-number for bit2?)
```

WMIevent(1006h) - WMI_REGDOMAIN_EVENT ;aka WMI_REG_DOMAIN_EVENT
Event Data (04h bytes):

```
  00h  A_UINT32 4   regDomain  ;80000188h on DSi (after firmware upload)
```

New Regulatory Domain Event.
Value 80000188h on DSi/AR6002 and 80000348h on DSi/AR6013 (these are based on the wifi firmware's "RGBD" DataSet, with 0188h="JP" and 0348h="US").
The country code seems to be derived from I2C EEPROM[008h], and the database is used to translate the country code to enabled channels.

WMIevent(1008h) - WMI_NEIGHBOR_REPORT_EVENT
Event Data:
typedef PREPACK struct {

```
  00h  A_INT8              1     numberOfAps;
  01h  WMI_NEIGHBOR_INFO   N*7   neighbor[1];
```

WMI_NEIGHBOR_INFO structure:

```
    A_UINT8 6    bssid[ATH_MAC_LEN];
    A_UINT8 1    bssFlags;            /* see WMI_BSS_FLAGS
```

WMI_BSS_FLAGS values:

```
  WMI_DEFAULT_BSS_FLAGS   = 00h
  WMI_PREAUTH_CAPABLE_BSS = 01h
  WMI_PMKID_VALID_BSS     = 02h
```

The WMI_NEIGHBOR_REPORT Event is generated by the target to inform the host of BSS's it has found that matches the current profile.
It can be used by the host to cache PMKs and/to initiate pre-authentication if the BSS supports it. The first bssid is always the current associated BSS.
The bssid and bssFlags information repeats according to the number or APs reported.

WMIevent(100Eh) - WMI_OPT_RX_FRAME_EVENT aka WMI_OPT_RX_INFO
Event Data (10h+N bytes):

```
  00h  A_UINT16 2   channel                        ;\
  02h  A_UINT8  1   frameType  ;see WMI_OPT_FTYPE  ; special frame info header
  03h  A_INT8   1   snr                            ;
  04h  A_UINT8  6   srcAddr[ATH_MAC_LEN]           ;
  0Ah  A_UINT8  6   bssid[ATH_MAC_LEN]             ;/
  10h  ...      ..  body (having WHAT length?)     ;-special frame body
```

Special frame receive Event.
Mechanism used to inform host of the receiption of the special frames.
The 802.11 header is not included.
See also: WMI_OPT_TX_FRAME_CMD

WMIevent(1010h) - WMI_EXTENSION_EVENT ;prefix for WMIX events...
Unknown, probably similar prefix as "WMIX_CMD_HDR" with 32bit WMIX value.

WMIevent(1010h:3007h) - WMIX_HB_CHALLENGE_RESP_EVENT ;used (reply to 2008h)
Event Data:
```
  00h  A_UINT32  4    cookie;   ;\same reply-format as command parameters
  04h  A_UINT32  4    source;   ;/
```
Actually, there are TWO or THREE more 32bit words appended on DSi... in relation to flags in MBOX0[04h], or actually MBOX0[04h] is the LENGTH in bytes of that extra stuff?!?!!

WMIevent(1011h) - WMI_CAC_EVENT
Supposedly related to Call Admission Control (CAC), eg. for VoIP.
Event Data (42h bytes):
```
  00h  A_UINT8  1    ac;
  01h  A_UINT8  1    cac_indication;
  02h  A_UINT8  1    statusCode;
  03h  A_UINT8  3Fh  tspecSuggestion[WMM_TSPEC_IE_LEN];
```
CAC_INDICATION values:
```
  CAC_INDICATION_ADMISSION      = 00h
  CAC_INDICATION_ADMISSION_RESP = 01h
  CAC_INDICATION_DELETE         = 02h
  CAC_INDICATION_NO_RESP        = 03h
```
#define WMM_TSPEC_IE_LEN = 63

WMIevent(1017h) - WMI_APLIST_EVENT
Event Data:
```
  00h  A_UINT8      1    apListVer;
  01h  A_UINT8      1    numAP;
  02h  WMI_AP_INFO  N*8  apList[1];
```
APLIST_VER values (only one defined):
```
    APLIST_VER1 = 1,
```
WMI_AP_INFO structure:
```
  typedef PREPACK union {
    WMI_AP_INFO_V1  8   apInfoV1;
  } POSTPACK WMI_AP_INFO;
```
WMI_AP_INFO_V1 structure:
```
    A_UINT8  6   bssid[ATH_MAC_LEN];
    A_UINT16 2   channel;
```

WMIevent(1019h) - WMI_GET_PMKID_LIST_EVENT ;reply to CMD F006h
See WMI_GET_PMKID_LIST_CMD for response details.

DSi Atheros Wifi - WMI Connect Functions

WMIcmd(0001h) - WMI_CONNECT_CMD
Parameters (34h bytes):
```
  00h  A_UINT8  1   networkType          ;somewhat NETWORK_TYPE related ?   (1)
  01h  A_UINT8  1   dot11AuthMode        ;aka DOT11_AUTH_MODE ? (1=Open, 2=WEP)
  02h  A_UINT8  1   authMode             ;aka AUTH_MODE ?                   (1)
  03h  A_UINT8  1   pairwiseCryptoType   ;aka CRYPTO_TYPE       (1=Open, 2=WEP)
  04h  A_UINT8  1   pairwiseCryptoLen                                       (0)
  05h  A_UINT8  1   groupCryptoType      ;aka CRYPTO_TYPE       (1=Open, 2=WEP)
  06h  A_UINT8  1   groupCryptoLen                                          (0)
  07h  A_UINT8  1   ssidLength
  08h  A_UCHAR  32  ssid[WMI_MAX_SSID_LEN]
  28h  A_UINT16 2   channel              ;in MHz
```

```
  2Ah  A_UINT8  6   bssid[ATH_MAC_LEN]
  30h  A_UINT32 4   ctrl_flags                                              (0)
```
Returns WMI_CONNECT_EVENT (or WMI_DISCONNECT_EVENT, eg. upon bad WEP password).
NETWORK_TYPE values:
```
  INFRA_NETWORK       = 01h      ;DSi uses 01h for Open/WEP/WPA/WPA2
  ADHOC_NETWORK       = 02h
  ADHOC_CREATOR       = 04h
  AP_NETWORK          = 10h
```
NETWORK_SUBTYPE values (unknown purpose, and unknown if they start at 0 or 1 or so):
```
  SUBTYPE_NONE        = unknown (maybe 0 or 1 or so?)
  SUBTYPE_BT          = SUBTYPE_NONE+1
  SUBTYPE_P2PDEV      = SUBTYPE_NONE+2
  SUBTYPE_P2PCLIENT   = SUBTYPE_NONE+3
  SUBTYPE_P2PGO       = SUBTYPE_NONE+4
```
DOT11_AUTH_MODE values:
```
  OPEN_AUTH           = 01h      ;DSi uses 01h for Open/WPA/WPA2
  SHARED_AUTH         = 02h      ;DSi uses 02h for WEP
  LEAP_AUTH           = 04h  /* different from IEEE_AUTH_MODE definitions
```
AUTH_MODE values:
```
  WMI_NONE_AUTH       = 01h      ;DSi uses 01h for Open/WEP
  WMI_WPA_AUTH        = 02h  ;\whatever maybe for RADIUS?
  WMI_WPA2_AUTH       = 04h  ;/
  WMI_WPA_PSK_AUTH    = 08h      ;DSi uses 03h (not 08h) for WPA-PSK
  WMI_WPA2_PSK_AUTH   = 10h      ;DSi uses 05h (not 10h) for WPA2-PSK
  WMI_WPA_AUTH_CCKM   = 20h  ;\whatever for "Cisco Centralized Key Management"?
  WMI_WPA2_AUTH_CCKM  = 40h  ;/
```
CRYPTO_TYPE values:
```
  NONE_CRYPT          = 01h      ;DSi uses 01h for Open
  WEP_CRYPT           = 02h      ;DSi uses 02h for WEP
  TKIP_CRYPT          = 04h      ;DSi uses 03h (not 04h) for WPA
  AES_CRYPT           = 08h      ;DSi uses 04h (not 08h) for WPA2
  WAPI_CRYPT          = 10h  ;only if WAPI_ENABLE
```
connect "ctrl_flags":
```
  0   CONNECT_ASSOC_POLICY_USER           = 0001h
  1   CONNECT_SEND_REASSOC                = 0002h
  2   CONNECT_IGNORE_WPAx_GROUP_CIPHER    = 0004h
  3   CONNECT_PROFILE_MATCH_DONE          = 0008h
  4   CONNECT_IGNORE_AAC_BEACON           = 0010h
  5   CONNECT_CSA_FOLLOW_BSS              = 0020h
  6   CONNECT_PYXIS_REMOTE                = 0040h  ;-old code from 2008
  6   CONNECT_DO_WPA_OFFLOAD              = 0040h  ;\
  7   CONNECT_DO_NOT_DEAUTH               = 0080h  ; new code from 2010
  8   CONNECT_WPS_FLAG                    = 0100h  ;
  9   CONNECT_IGNORE_BSSID_HINT           = 0200h  ;
  16  AP_NO_DISASSOC_UPON_DEAUTH          = 10000h ;/ <--AP configuration flags
```
DEFAULT_CONNECT_CTRL_FLAGS = (CONNECT_CSA_FOLLOW_BSS)

WMIcmd(0002h) - WMI_RECONNECT_CMD
Parameters (optional...?... 08h bytes):
```
  00h  A_UINT16 2   channel                   ;hint
  02h  A_UINT8  6   bssid[ATH_MAC_LEN]        ;mandatory if set
```

WMIcmd(0003h) - WMI_DISCONNECT_CMD
Parameters:
```
  Unknown (none?)
```
Returns WMI_DISCONNECT_EVENT. Uhm, or seems to crash on AR6013...?

WMIcmd(000Dh) - WMI_SET_DISC_TIMEOUT_CMD ;aka WMI_SET_DISCONNECT_TIMEOUT
Parameters (01h bytes):
```
  00h  A_UINT8  1   disconnectTimeout         ;seconds
```

WMIevent(1002h) - WMI_CONNECT_EVENT

First 10h bytes of response can be "infra_ibss_bss" or "ap_sta" or "ap_bss" (unclear which selects/indicates which it is, maybe the parameters from preceeding WMI_CONNECT_CMD, or from WMI_BSSINFO_EVENT's frame body?)

Also unknown what is sent in case of connect ERROR (maybe DISCONNECT_EVENT?)

Event Data:

```
 When "infra_ibss_bss":                      ;<-- occurs for my WPA2 connect
  00h  A_UINT16 2   channel                     ;in MHz
  02h  A_UINT8  6   bssid[ATH_MAC_LEN]          ;
  08h  A_UINT16 2   listenInterval              ;0064h
  0Ah  A_UINT16 2   beaconInterval              ;0064h
  0Ch  A_UINT32 4   networkType                 ;00000001h
 When "ap_sta":
  00h  A_UINT8  1   phymode
  01h  A_UINT8  1   aid
  02h  A_UINT8  6   mac_addr[ATH_MAC_LEN]
  08h  A_UINT8  1   auth
  09h  A_UINT8  1   keymgmt
  0Ah  A_UINT16 2   cipher
  0Ch  A_UINT8  1   apsd_info
  0Dh  A_UINT8  3   unused[3]
 When "ap_bss":
  00h  A_UINT16 2   channel
  02h  A_UINT8  6   bssid[ATH_MAC_LEN]
  08h  A_UINT8  8   unused[8]
 And, in all three cases:
  10h  A_UINT8  1   beaconIeLen                 ;16h
  11h  A_UINT8  1   assocReqLen                 ;2Fh
  12h  A_UINT8  1   assocRespLen                ;16h
  13h  A_UINT8  ..  assocInfo[1]                ;whatever 100 bytes?
```

In STA mode networkType comes along with connected phy mode.

To get networkType, WMI_NETWORK_TYPE (networkType).

To get connected phymode, WMI_CONNECTED_PHYMODE(networkType) will give the phymode value.

WMIevent(1003h) - WMI_DISCONNECT_EVENT

This event can occur when using WMI_CONNECT with wrong WEP password (instead of WMI_CONNECT_EVENT), or when failing to respond to WPA/WPA2 handshake messages, or when using WMI_DISCONNECT.

Event Data (0Ah+N bytes):

```
  00h  A_UINT16 2    protocolReasonStatus ;reason code, see 802.11 spec.
  02h  A_UINT8  6    bssid[ATH_MAC_LEN]   ;set if known
  08h  A_UINT8  1    disconnectReason     ;see WMI_DISCONNECT_REASON
  09h  A_UINT8  1    assocRespLen (00h=none)
  0Ah  A_UINT8  N*1  assocInfo[1]
```

WMI_DISCONNECT_REASON values:

```
  NO_NETWORK_AVAIL      = 01h  ;-occurs on wrong WEP key
  LOST_LINK             = 02h  ;-bmiss
  DISCONNECT_CMD        = 03h  ;-requested via disconnect command
  BSS_DISCONNECTED      = 04h  ;-occurs after some minutes of inactivity?
  AUTH_FAILED           = 05h  ;-reportedly occurs if AP was already connected?
  ASSOC_FAILED          = 06h
  NO_RESOURCES_AVAIL    = 07h
  CSERV_DISCONNECT      = 08h
  INVALID_PROFILE       = 0Ah
  DOT11H_CHANNEL_SWITCH = 0Bh
  PROFILE_MISMATCH      = 0Ch
  PYXIS_VIRT_ADHOC_DISC = 0Dh  ;-old code from 2008
  CONNECTION_EVICTED    = 0Dh  ;\
  IBSS_MERGE            = 0Eh  ; new code from 2010
  EXCESS_TX_RETRY       = 0Fh  ;/  <--TX frames failed after excessive retries
```

DSi Atheros Wifi - WMI Channel and Cipher Functions

WMIcmd(000Eh) - WMI_GET_CHANNEL_LIST_CMD ;reply 000Eh ;aka WMI_CHANNEL_LIST

Parameters:

```
None
```

WMI_GET_CHANNEL_LIST_CMD reply ;aka WMI_CHANNEL_LIST_REPLY

```
00h  A_UINT8  1    reserved1          ;whatever (zero)
01h  A_UINT8  1    numChannels        ;number of channels in reply (N)
02h  A_UINT16 N*2  channelList[1]     ;channels in MHz
02h+N*2       2    Zero (0000h)       ;undocumented (end of list or so)
```

Reply size is 1Ah bytes on DSi (2+(11*2)+2 bytes, for 11 channels, plus ending 0000h; that ending halfward isn't officially mentioned to exist).

WMIcmd(0011h) - WMI_SET_CHANNEL_PARAMS_CMD ;aka WMI_CHANNEL_PARAMS_CMD

Parameters (04h+N*2 bytes):

```
00h  A_UINT8  1   reserved1           ;whatever (?)
01h  A_UINT8  1   scanParam           ;set if enable scan
02h  A_UINT8  1   phyMode             ;see WMI_PHY_MODE
03h  A_UINT8  1   numChannels         ;how many channels follow
04h  A_UINT16 N*2 channelList[1]      ;channels in MHz
```

WMI_PHY_MODE values:

```
WMI_11A_MODE     = 01h
WMI_11G_MODE     = 02h
WMI_11AG_MODE    = 03h
WMI_11B_MODE     = 04h
WMI_11GONLY_MODE = 05h
```

#define WMI_MAX_CHANNELS = 32

Some (or all?) DSi's have only channel 1-11 enabled, and throw WMI_CMDERROR_EVENT when trying to set channel 12,13,14. Unknown what defines which channels are enabled (probably something in EEPROM). The enabled channels can be obtained via WMI_GET_CHANNEL_LIST_CMD (which is probably related to the country reported in WMI_REGDOMAIN_EVENT).

WMIevent(101Ah) - WMI_CHANNEL_CHANGE_EVENT ;<-- used on DSi ?

Event Data (06h bytes):

```
00h  A_UINT16  2  oldChannel;   ;\uh, old is 16bit, new is misaligned 32bit?
02h  A_UINT32  4  newChannel;   ;/(DSi does really send 6 bytes)
```

This event is not defined in "AR6kSDK.build_sw.18", however, "101Ah" is USED on DSi.

WMIcmd(0016h) - WMI_ADD_CIPHER_KEY_CMD

Parameters (2Dh bytes on DSi?, but other sources claim 2Ch or 33h bytes):

```
2Ch 2Dh 33h   <---- total size (on DSi it's 2Dh, ie. middle column)
00h 00h 00h  A_UINT8  1   keyIndex ;aka WMI_MAX_KEY_INDEX ?
01h 01h 01h  A_UINT8  1   keyType  ;aka CRYPTO_TYPE
02h 02h 02h  A_UINT8  1   keyUsage      ;KEY_USAGE
03h 03h 03h  A_UINT8  1   keyLength
04h 04h 04h  A_UINT8  8   keyRSC[8]     ;key replay sequence counter
0Ch 0Ch 0Ch  A_UINT8  32  key[WMI_MAX_KEY_LEN]    ;aka password
--  2Ch 2Ch  A_UINT8  1   key_op_ctrl   ;Additional Key Control information
--  --  2Dh  A_UINT8  6   key_macaddr[ATH_MAC_LEN]
```

KEY_USAGE values:

```
PAIRWISE_USAGE     = 00h ;<-- DSi browser uses THIS for WPA/WPA2 key 0
GROUP_USAGE        = 01h ;<-- DSi browser uses THIS for WEP/WPA/WPA2 key 1..3
TX_USAGE           = 02h ;<-- reportedly "default Tx Key - Static WEP only"
Undoc (or 01h+02h) = 03h ;<-- DSi browser uses THIS for WEP key 0
```

Bit Flag. (aka key_op_ctrl values?):

```
Bit 0 - Initialise TSC - default is Initialize
KEY_OP_INIT_TSC    = 01h
```

```
KEY_OP_INIT_RSC    = 02h
KEY_OP_INIT_WAPIPN = 10h (only if "WAPI_ENABLE")
KEY_OP_INIT_VAL    = 03h  ;<-- Default Initialise the TSC & RSC ;used by DSi
KEY_OP_VALID_MASK  = 03h
```

More constants:

```
WMI_MIN_KEY_INDEX = 0
WMI_MAX_KEY_INDEX = 3  ;<-- when not "WAPI_ENABLE"
WMI_MAX_KEY_INDEX = 7  ;<-- when "WAPI_ENABLE" (wapi grpKey 0-3, prwKey 4-7)
WMI_MAX_KEY_LEN   = 32
```

WEP Cipher Keys

All four keys (with KeyIndex=0..3) must be set before WMI_CONNECT_CMD. The NDS/DSi user interface allows to define only one WEP key 0, but one must add dummy keys for key 1..3, else connect will fail.

WPA/WPA2 Cipher Keys

For WPA/WPA2, WMI_CONNECT_CMD isn't working fully automated: After WMI_CONNECT_EVENT, one must manually receive & reply EAPOL handshake messages, and then add the cipher keys based on the message contents. The used keys are key 0 (for pairwise cipher), and key 1 or 2 (for the current group cipher key).

DS Wifi WPA/WPA2 Handshake Messages (EAPOL)
DS Wifi WPA/WPA2 Keys and MICs
DS Wifi WPA/WPA2 Encryption
DS Firmware Wifi Calibration Data
DS Firmware Wifi Internet Access Points

Moreover, the access point may throw further EAPOL messages for assigning new group keys every once and then, and one must also manually apply that new key via ADD_CIPHER (the group key index switches between 1 and 2 on each update).

```
KeyIndex=0,   key=PTK[20h..2Fh]+PTK[38h..3Fh]+PTK[30h..37h], RSC=0
KeyIndex=1/2, key=GTK[00h..0Fh]+PTK[18h..1Fh]+PTK[10h..17h], RSC=EAPOL RSC
```

WAPI Cipher Keys

Some atheros firmwares have "WAPI_ENABLE" with eight key indices (grpKey=0..3, prwKey=4..7). The DSi firmware probably doesn't have that WAPI stuff.

WMIcmd(0017h) - WMI_DELETE_CIPHER_KEY_CMD ;ignored dummy command on DSi

Parameters (01h bytes):

```
00h  A_UINT8  1   keyIndex
```

WMIcmd(0018h) - WMI_ADD_KRK_CMD ;ignored dummy command on DSi

Parameters (10h bytes):

```
00h  A_UINT8  16  krk[WMI_KRK_LEN]
```

#define WMI_KRK_LEN = 16

KRK maybe means "Key Registration with Knowledge"?

WMIcmd(0019h) - WMI_DELETE_KRK_CMD ;ignored dummy command on DSi

Parameters:

```
Unknown (none?) (or maybe same as for ADD_KRK ?) (seems to be NONE on DSi)
```

WMIcmd(0020h) - WMI_SET_TKIP_COUNTERMEASURES_CMD

Parameters (01h bytes):

```
00h  A_UINT8  1  cm_en                      ;WMI_TKIP_CM_CONTROL
```

WMI_TKIP_CM_CONTROL values:

```
WMI_TKIP_CM_DISABLE = 00h
WMI_TKIP_CM_ENABLE  = 01h
```

Note: There are also "CM_CONNECT_TYPE" values in "cnxmgmt.h", is THAT related?

WMIevent(1009h) - WMI_TKIP_MICERR_EVENT

Event Data (02h bytes):

```
00h  A_UINT8  1  keyid
01h  A_UINT8  1  ismcast
```

DSi Atheros Wifi - WMI Scan Functions

WMIcmd(0007h) - WMI_START_SCAN_CMD

Parameters (theoretically 12h+N*2 bytes, but 14h bytes when N=0 ?):

```
  00h  A_BOOL   4   forceFgScan
  04h  A_BOOL   4   isLegacy            For Legacy Cisco AP compatibility
  08h  A_UINT32 4   homeDwellTime       Max duration in the home channel (msec)
  0Ch  A_UINT32 4   forceScanInterval   Time interval between scans (msec)
  10h  A_UINT8  1   scanType            WMI_SCAN_TYPE
  11h  A_UINT8  1   numChannels         how many channels follow
  12h  A_UINT16 N*2 channelList[1]      channels in MHz
```

WMI_SCAN_TYPE values:

```
  WMI_LONG_SCAN  = 0
  WMI_SHORT_SCAN = 1
```

Old code from 2008 did (additionally) use value 0 and 1 as so:

```
  WMI_PYXIS_PAS_DSCVR = 0
  WMI_PYXIS_ACT_DSCVR = 1
```

The DSi Browser receives one or more WMI_BSSINFO_EVENT's (if there are any beacons on the current channel), and, once when the DwellTime (mul5?) has ellapsed, finally receives WMI_SCAN_COMPLETE_EVENT.

WMIcmd(0008h) - WMI_SET_SCAN_PARAMS_CMD

Parameters (14h bytes):

```
  00h  A_UINT16 2  fg_start_period         ;seconds
  02h  A_UINT16 2  fg_end_period           ;seconds
  04h  A_UINT16 2  bg_period               ;seconds
  06h  A_UINT16 2  maxact_chdwell_time     ;msec
  08h  A_UINT16 2  pas_chdwell_time        ;msec
  0Ah  A_UINT8  1  shortScanRatio          ;how many shorts scan for one long
  0Bh  A_UINT8  1  scanCtrlFlags
  0Ch  A_UINT16 2  minact_chdwell_time     ;msec
  0Eh  A_UINT16 2  maxact_scan_per_ssid    ;max active scans per ssid
  10h  A_UINT32 4  max_dfsch_act_time      ;msec
```

#define WMI_SHORTSCANRATIO_DEFAULT = 3

Warning: ScanCtrlFlag value of FFh is used to disable all flags in WMI_SCAN_PARAMS_CMD

Do not add any more flags to WMI_SCAN_CTRL_FLAG_BITS

WMI_SCAN_CTRL_FLAGS_BITS:

```
  CONNECT_SCAN_CTRL_FLAGS   = 01h  ;set if can scan in the Connect cmd
  SCAN_CONNECTED_CTRL_FLAGS = 02h  ;set if scan for the SSID it is
                                   ;  already connected to
  ACTIVE_SCAN_CTRL_FLAGS    = 04h  ;set if enable active scan
  ROAM_SCAN_CTRL_FLAGS      = 08h  ;set if enable roam scan when bmiss
                                   ;  and lowrssi
  REPORT_BSSINFO_CTRL_FLAGS = 10h  ;set if follows customer BSSINFO
                                   ;  reporting rule
  ENABLE_AUTO_CTRL_FLAGS    = 20h  ;if disabled, target doesn't
                                   ;  scan after a disconnect event
  ENABLE_SCAN_ABORT_EVENT   = 40h  ;Scan complete event with canceled status
                                   ;  will be generated when a scan is
                                   ;  prempted before it gets completed
 #define CAN_SCAN_IN_CONNECT(flags)      (flags & CONNECT_SCAN_CTRL_FLAGS)
 #define CAN_SCAN_CONNECTED(flags)       (flags & SCAN_CONNECTED_CTRL_FLAGS)
 #define ENABLE_ACTIVE_SCAN(flags)       (flags & ACTIVE_SCAN_CTRL_FLAGS)
 #define ENABLE_ROAM_SCAN(flags)         (flags & ROAM_SCAN_CTRL_FLAGS)
 #define CONFIG_REPORT_BSSINFO(flags)    (flags & REPORT_BSSINFO_CTRL_FLAGS)
 #define IS_AUTO_SCAN_ENABLED(flags)     (flags & ENABLE_AUTO_CTRL_FLAGS)
 #define SCAN_ABORT_EVENT_ENABLED(flags) (flags & ENABLE_SCAN_ABORT_EVENT)
```

#define DEFAULT_SCAN_CTRL_FLAGS = (CONNECT_SCAN_CTRL_FLAGS | SCAN_CONNECTED_CTRL_FLAGS | ACTIVE_SCAN_CTRL_FLAGS | ROAM_SCAN_CTRL_FLAGS | ENABLE_AUTO_CTRL_FLAGS)

WMIevent(100Ah) - WMI_SCAN_COMPLETE_EVENT
Event Data (old: None, or new: status, 04h bytes):
```
  00h  A_INT32  4  status;    ;aka "staus"
```
(whatever, usually/always zero)
(can be 10h... when already connected maybe?)

Note: There are several "SCAN_xxx" and "xxx_SCPRI" values defined in "discovery.h" - purpose is unknown (maybe that stuff is used only internally).

DSi Atheros Wifi - WMI Bit Rate Functions

WMIcmd(0034h) - WMI_SET_FIXRATES_CMD ;aka WMI_FIX_RATES_CMD
Parameters (02h bytes on DSi?, but other sources claim 08h bytes):
```
  02h 08h   <---- total size (on DSi it's 02h, ie. left column)
  00h --  A_UINT16 2   fixRateMask     ;0..0FFFh          ;see WMI_BIT_RATE
  --  00h A_UINT32 4   fixRateMask(0)  ;0..0FFFFFFFh      ;see WMI_BIT_RATE
  --  04h A_UINT32 4   fixRateMask(1)  ;0..0FFFFFFFh      ;see WMI_BIT_RATE
```
WMIcmd(0035h) - WMI_GET_FIXRATES_CMD ;reply 0035h
Parameters:
```
  Unknown (none?)
```
Reply: WMI_GET_FIXRATES aka WMI_FIX_RATES_REPLY
Event Data (02h bytes on DSi?, but other sources claim 08h bytes):
```
  02h 08h   <---- total size (on DSi it's 02h, ie. left column)
  00h --  A_UINT16 2   fixRateMask     ;0..0FFFh           ;see WMI_BIT_RATE
  --  00h A_UINT32 4   fixRateMask(0)  ;0..0FFFFFFFh       ;see WMI_BIT_RATE
  --  04h A_UINT32 4   fixRateMask(1)  ;0..0FFFFFFFh       ;see WMI_BIT_RATE
```
See "FIX_RATE_xxx" values.

WMIcmd(0048h or 0047h) - WMI_SET_FRAMERATES_CMD ;aka WMI_FRAME_RATES_CMD
DSi uses command number 0048h for this (whilst most or all official Atheros code is using number 0047h).
Parameters (04h bytes, or 0Ch bytes on hardware with more rates):
```
  04h 0Ch   <---- total size (on DSi it's 04h, ie. left column)
  00h 00h  A_UINT8  1   bEnableMask               (used: 01h) ;whatever?
  01h 01h  A_UINT8  1   frameType  (D4h=special?) (used: A4h) ;type and subtype
  02h --   A_UINT16 2   frameRateMask ;0..0FFFh (used: FFF7h) ;see WMI_BIT_RATE
  --  02h  A_UINT8  2   reserved[2]                           ;for alignment
  --  04h  A_UINT32 4   frameRateMask(0)  ;0..0FFFFFFFh       ;see WMI_BIT_RATE
  --  08h  A_UINT32 4   frameRateMask(1)  ;0..0FFFFFFFh       ;see WMI_BIT_RATE
```
Note: FRAMERATE is apparently meant to be "bitrate" for certain "packet types" (frames). The type and subtype values are maybe meant to resemble those in Frame Control (FC) field, or the FTYPE_xxx constants in some atheros source code?
Reportedly there is a "WMI_FRAME_RATES_REPLY" with same Reply structure as above Parameter structure, the DSi doesn't support that feature (and it's rather unknown if/when/how that Reply would be sent if the feature were implemented... maybe as event, or as reply to SET_FRAMERATES, or as reply to whatever other cmd).

WMIcmd(F000h) - WMI_SET_BITRATE_CMD ;aka WMI_BIT_RATE_CMD
Parameters (03h bytes):
```
  00h  A_INT8  1   rateIndex      (FFh=Auto)     ;see WMI_BIT_RATE
  01h  A_INT8  1   mgmtRateIndex  (00h=1Mbit/s)
  02h  A_INT8  1   ctlRateIndex   (00h=1Mbit/s)
```

Note: This doesn't really work. Using rateIndex=FFh (or 0Bh) does currently reach only max 600Kbyte/s upload speed (which MIGHT be a bug in my software). However, using rateIndex=00h is slower, but still reaches more than 200Kbyte/s (which is outright wrong and impossible at 1Mbit/s).

WMIcmd(F001h) - WMI_GET_BITRATE_CMD ;reply F001h

Parameters:

```
Unknown (none?)
```

Reply:

```
00h  A_INT8   1    rateIndex           ;see WMI_BIT_RATE
```

WMI_BIT_RATE values

```
Mbit/s        = Index      Mbit/s            = Mask
RATE_AUTO     = -1         -                 = -
RATE_1Mb      = 0          FIX_RATE_1Mb      = 1h
RATE_2Mb      = 1          FIX_RATE_2Mb      = 2h
RATE_5_5Mb    = 2          FIX_RATE_5_5Mb    = 4h
RATE_11Mb     = 3          FIX_RATE_11Mb     = 8h
RATE_6Mb      = 4          FIX_RATE_6Mb      = 10h
RATE_9Mb      = 5          FIX_RATE_9Mb      = 20h
RATE_12Mb     = 6          FIX_RATE_12Mb     = 40h
RATE_18Mb     = 7          FIX_RATE_18Mb     = 80h
RATE_24Mb     = 8          FIX_RATE_24Mb     = 100h
RATE_36Mb     = 9          FIX_RATE_36Mb     = 200h
RATE_48Mb     = 10         FIX_RATE_48Mb     = 400h
RATE_54Mb     = 11         FIX_RATE_54Mb     = 800h
```

Below only for newer (non-DSi) revisions (with 32bit RATE_MASK values):

```
RATE_MCS_0_20 = 12         FIX_RATE_MCS_0_20 = 1000h
RATE_MCS_1_20 = 13         FIX_RATE_MCS_1_20 = 2000h
RATE_MCS_2_20 = 14         FIX_RATE_MCS_2_20 = 4000h
RATE_MCS_3_20 = 15         FIX_RATE_MCS_3_20 = 8000h
RATE_MCS_4_20 = 16         FIX_RATE_MCS_4_20 = 10000h
RATE_MCS_5_20 = 17         FIX_RATE_MCS_5_20 = 20000h
RATE_MCS_6_20 = 18         FIX_RATE_MCS_6_20 = 40000h
RATE_MCS_7_20 = 19         FIX_RATE_MCS_7_20 = 80000h
RATE_MCS_0_40 = 20         FIX_RATE_MCS_0_40 = 100000h
RATE_MCS_1_40 = 21         FIX_RATE_MCS_1_40 = 200000h
RATE_MCS_2_40 = 22         FIX_RATE_MCS_2_40 = 400000h
RATE_MCS_3_40 = 23         FIX_RATE_MCS_3_40 = 800000h
RATE_MCS_4_40 = 24         FIX_RATE_MCS_4_40 = 1000000h
RATE_MCS_5_40 = 25         FIX_RATE_MCS_5_40 = 2000000h
RATE_MCS_6_40 = 26         FIX_RATE_MCS_6_40 = 4000000h
RATE_MCS_7_40 = 27         FIX_RATE_MCS_7_40 = 8000000h
```

Get bit rate cmd uses same definition as set bit rate cmd

DSi Atheros Wifi - WMI Threshold Functions

Below commands allow to define thresholds for RSSI, SNR, LQ.

```
RSSI  Received Signal Strength Indicator
SNR   Signal to Noise Ratio
LQ    Link Quality
```

WMIcmd(0021h) - WMI_RSSI_THRESHOLD_PARAMS_CMD

Parameters (20h bytes):

```
00h  A_UINT32 4  pollTime                ;Polling time as a factor of LI
04h  A_INT16  2  thresholdAbove1_Val     ;lowest of upper
06h  A_INT16  2  thresholdAbove2_Val
08h  A_INT16  2  thresholdAbove3_Val
0Ah  A_INT16  2  thresholdAbove4_Val
0Ch  A_INT16  2  thresholdAbove5_Val
```

```
  0Eh  A_INT16  2  thresholdAbove6_Val     ;highest of upper
  10h  A_INT16  2  thresholdBelow1_Val     ;lowest of bellow
  12h  A_INT16  2  thresholdBelow2_Val
  14h  A_INT16  2  thresholdBelow3_Val
  16h  A_INT16  2  thresholdBelow4_Val
  18h  A_INT16  2  thresholdBelow5_Val
  1Ah  A_INT16  2  thresholdBelow6_Val     ;highest of bellow
  1Ch  A_UINT8  1  weight                  ;"alpha"
  1Dh  A_UINT8  3  reserved[3]
 Setting the polltime to 0 would disable polling.
 Threshold values are in the ascending order, and should agree to:
   (lowThreshold_lowerVal < lowThreshold_upperVal < highThreshold_lowerVal
     < highThreshold_upperVal)
```
See also: WMI_RSSI_THRESHOLD_EVENT

WMIcmd(002Fh) - WMI_SNR_THRESHOLD_PARAMS_CMD

Parameters (10h bytes):
```
  00h  A_UINT32 4  pollTime                ;Polling time as a factor of LI
  04h  A_UINT8  1  weight                  ;"alpha"
  05h  A_UINT8  1  thresholdAbove1_Val     ;lowest of uppper   ;uh, ppper?
  06h  A_UINT8  1  thresholdAbove2_Val
  07h  A_UINT8  1  thresholdAbove3_Val
  08h  A_UINT8  1  thresholdAbove4_Val     ;highest of upper
  09h  A_UINT8  1  thresholdBelow1_Val     ;lowest of bellow   ;uh bell?
  0Ah  A_UINT8  1  thresholdBelow2_Val
  0Bh  A_UINT8  1  thresholdBelow3_Val
  0Ch  A_UINT8  1  thresholdBelow4_Val     ;highest of bellow  ;uh bell?
  0Dh  A_UINT8  3  reserved[3]
```
Setting the polltime to 0 would disable polling.
See also: WMI_SNR_THRESHOLD_EVENT

WMIcmd(0030h) - WMI_LQ_THRESHOLD_PARAMS_CMD

Parameters (0Ch bytes):
```
  00h  A_UINT8  1  enable              ;<-- enable (unlike SNR command)
  01h  A_UINT8  1  thresholdAbove1_Val   ;\these parameters seem to be same as
  02h  A_UINT8  1  thresholdAbove2_Val   ; for WMI_SNR_THRESHOLD_PARAMS_CMD
  03h  A_UINT8  1  thresholdAbove3_Val   ;
  04h  A_UINT8  1  thresholdAbove4_Val   ;
  05h  A_UINT8  1  thresholdBelow1_Val   ;
  06h  A_UINT8  1  thresholdBelow2_Val   ;
  07h  A_UINT8  1  thresholdBelow3_Val   ;
  08h  A_UINT8  1  thresholdBelow4_Val   ;
  09h  A_UINT8  3  reserved[3]           ;/
```
See also: WMI_LQ_THRESHOLD_EVENT

WMIcmd(0033h) - WMI_CLR_RSSI_SNR_CMD

Parameters:
```
  Unknown (none?)
```
Probably somehow related to RSSI_THRESHOLD and SNR_THRESHOLD.

WMIevent(100Ch) - WMI_RSSI_THRESHOLD_EVENT

Event Data (03h bytes):
```
  00h  A_INT16  2   rssi;
  02h  A_UINT8  1   range;
```
WMI_RSSI_THRESHOLD_VAL values (which are probably meant to occurr in one of above fields, unclear which one though):
```
  WMI_RSSI_THRESHOLD1_ABOVE = 0
  WMI_RSSI_THRESHOLD2_ABOVE = 1
  WMI_RSSI_THRESHOLD3_ABOVE = 2
  WMI_RSSI_THRESHOLD4_ABOVE = 3
  WMI_RSSI_THRESHOLD5_ABOVE = 4
```

```
  WMI_RSSI_THRESHOLD6_ABOVE = 5
  WMI_RSSI_THRESHOLD1_BELOW = 6
  WMI_RSSI_THRESHOLD2_BELOW = 7
  WMI_RSSI_THRESHOLD3_BELOW = 8
  WMI_RSSI_THRESHOLD4_BELOW = 9
  WMI_RSSI_THRESHOLD5_BELOW = 10
  WMI_RSSI_THRESHOLD6_BELOW = 11
```
Indicate the RSSI events to host. Events are indicated when we breach a thresold value. (uh, how old do we breach?)

WMIevent(1012h) - WMI_SNR_THRESHOLD_EVENT
Event Data (02h bytes):
```
  00h  A_UINT8  1  range        ;WMI_SNR_THRESHOLD_VAL
  01h  A_UINT8  1  snr
```
WMI_SNR_THRESHOLD_VAL values:
```
  WMI_SNR_THRESHOLD1_ABOVE = 1
  WMI_SNR_THRESHOLD1_BELOW = 2
  WMI_SNR_THRESHOLD2_ABOVE = 3
  WMI_SNR_THRESHOLD2_BELOW = 4
  WMI_SNR_THRESHOLD3_ABOVE = 5
  WMI_SNR_THRESHOLD3_BELOW = 6
  WMI_SNR_THRESHOLD4_ABOVE = 7
  WMI_SNR_THRESHOLD4_BELOW = 8
```

WMIevent(1013h) - WMI_LQ_THRESHOLD_EVENT
Event Data (05h bytes):
```
  00h  A_INT32  4  lq
  04h  A_UINT8  1  range        ;WMI_LQ_THRESHOLD_VAL
```
WMI_LQ_THRESHOLD_VAL values:
```
  WMI_LQ_THRESHOLD1_ABOVE = 1
  WMI_LQ_THRESHOLD1_BELOW = 2
  WMI_LQ_THRESHOLD2_ABOVE = 3
  WMI_LQ_THRESHOLD2_BELOW = 4
  WMI_LQ_THRESHOLD3_ABOVE = 5
  WMI_LQ_THRESHOLD3_BELOW = 6
  WMI_LQ_THRESHOLD4_ABOVE = 7
  WMI_LQ_THRESHOLD4_BELOW = 8
```

DSi Atheros Wifi - WMI Error, Retry and Debug Functions

WMIcmd(0024h) - WMI_SET_RETRY_LIMITS_CMD
Parameters:
```
  00h  A_UINT8  1  frameType          ;WMI_FRAMETYPE
  01h  A_UINT8  1  trafficClass       ;applies only to DATA_FRAMETYPE
  02h  A_UINT8  1  maxRetries
  03h  A_UINT8  1  enableNotify
```
This command is used to customize the number of retries the wlan device will perform on a given frame.
```
  WMI_MIN_RETRIES = 2
  WMI_MAX_RETRIES = 13
```
WMI_FRAMETYPE values:
```
  MGMT_FRAMETYPE    = 0
  CONTROL_FRAMETYPE = 1
  DATA_FRAMETYPE    = 2
```

WMIcmd(0022h) - WMI_TARGET_ERROR_REPORT_BITMASK_CMD
Parameters (04h bytes):
```
  00h  A_UINT32 4   bitmask    ;... probably "WMI_TARGET_ERROR_VAL" ?
```
Sets the error reporting event bitmask in target. Target clears it upon an error. Subsequent errors are counted, but not reported via event, unless the bitmask is set again.

See also: WMI_TARGET_ERROR_REPORT_EVENT

WMIevent(1005h) - WMI_CMDERROR_EVENT ;aka WMI_CMD_ERROR_EVENT
Event Data (03h bytes):
```
  00h  A_UINT16 2   commandId    ;on DSi, this can be: 0001h,0011h,0016h,0026h
  02h  A_UINT8  1   errorCode    ;on DSi, this can be: 01h,02h
```
WMI_ERROR_CODE values:
```
  INVALID_PARAM  = 01h
  ILLEGAL_STATE  = 02h
  INTERNAL_ERROR = 03h
  DFS_CHANNEL    = 04h
```
Command Error Event

WMIevent(100Dh) - WMI_ERROR_REPORT_EVENT ;aka WMI_TARGET_ERROR_REPORT_EVENT
Event Data (04h bytes):
```
  00h  A_UINT32  4  errorVal
```
WMI_TARGET_ERROR_VAL values:
```
  WMI_TARGET_PM_ERR_FAIL      = 00000001h
  WMI_TARGET_KEY_NOT_FOUND    = 00000002h
  WMI_TARGET_DECRYPTION_ERR   = 00000004h
  WMI_TARGET_BMISS            = 00000008h
  WMI_PSDISABLE_NODE_JOIN     = 00000010h
  WMI_TARGET_COM_ERR          = 00000020h
  WMI_TARGET_FATAL_ERR        = 00000040h
  WMI_TARGET_BCN_FOUND        = 00000080h
```

WMIevent(1014h) - WMI_TX_RETRY_ERR_EVENT
Event Data (01h bytes):
```
  00h  A_UINT8  1   retrys
```

WMIcmd(002Eh:2009h) - WMIX_DBGLOG_CFG_MODULE_CMD
Parameters (said to be as so):
```
  00h  A_UINT32 4   cfgvalid        ;mask with valid config bits (uh, what?)
 When some case:
  04h  A_UINT32 4   dbglog_config   ;see "dbglog_config" description below
 When some other case:
  04h  A_UINT32 4   value           ;uh, what "value" (maybe alias for above?)
```
dbglog_config value:
```
  Bit0-15   DBGLOG_MODULE_LOG_ENABLE     ;logging enable flags for module 0-15
  Bit16     DBGLOG_REPORTING_ENABLED     ;reporting enable flag
  Bit17-19  DBGLOG_TIMESTAMP_RESOLUTION  ;timestamp resolution (default=1 ms)
  Bit20-29  DBGLOG_REPORT_SIZE           ;report size in number of messages
  Bit30-31  Reserved                     ;reserved
```
dbglog message value (with numeric "message" IDs):
```
  Bit0-15    DBGLOG_TIMESTAMP  ;contains bit8-23 of the LF0 timer (0..FFFFh)
  Bit16-25   DBGLOG_DBGID      ;minor ID (defined in "dbglog_id.h")
  Bit26-29   DBGLOG_MODULEID   ;major ID (defined in "dbglog.h")
  Bit30-31   DBGLOG_NUM_ARGS   ;whatever "num args"
```

WMIevent(1010h:3008h) - WMIX_DBGLOG_EVENT ;used (probably related to 2009h)
Event Data:
```
  Unknown (?) (probably related/enabled by WMIX_DBGLOG_CFG_MODULE_CMD)
```
DSi Browser does receive this - with LOTs of data (almost 1.5Kbyte)

DSi Atheros Wifi - WMI Priority Stream Functions

WMIcmd(0005h) - WMI_CREATE_PSTREAM_CMD ;aka WMI_CRE_PRIORITY_STREAM

Parameters (3Fh bytes in DSi, but other variants exist, too):

```
  old 3Fh 40h  <---- total size (on DSi it's 3Fh, ie. middle column)
  0Bh 00h 00h  A_UINT32 4  minServiceInt      ;in msec (14h=20)
  0Fh 04h 04h  A_UINT32 4  maxServiceInt      ;in msec (14h=20)
  13h 08h 08h  A_UINT32 4  inactivityInt      ;in msec (98967Fh=9999999)
  17h 0Ch 0Ch  A_UINT32 4  suspensionInt      ;in msec (-1)
  1Bh 10h 10h  A_UINT32 4  serviceStartTime             (0)
  1Fh 14h 14h  A_UINT32 4  minDataRate        ;in bps  (14500h=83200)
  23h 18h 18h  A_UINT32 4  meanDataRate       ;in bps  (14500h=83200)
  27h 1Ch 1Ch  A_UINT32 4  peakDataRate       ;in bps  (14500h=83200)
  2Bh 20h 20h  A_UINT32 4  maxBurstSize               (0)
  2Fh 24h 24h  A_UINT32 4  delayBound                 (0)
  33h 28h 28h  A_UINT32 4  minPhyRate         ;in bps  (5B8D80h=6000000)
  37h 2Ch 2Ch  A_UINT32 4  sba                        (2000h=8192)
  3Bh 30h 30h  A_UINT32 4  mediumTime                 (0)
  07h 34h 34h  A_UINT16 2  nominalMSDU        ;in octeCts   (80D0h=?)
  09h 36h 36h  A_UINT16 2  maxMSDU            ;in octeCts   (00D0h=?)
  00h 38h 38h  A_UINT8  1  trafficClass                 (00h)
  01h 39h 39h  A_UINT8  1  trafficDirection   ;DIR_TYPE     (02h=Bidir)
  02h 3Ah 3Ah  A_UINT8  1  rxQueueNum                   (FFh)
  03h 3Bh 3Bh  A_UINT8  1  trafficType        ;TRAFFIC_TYPE (01h=Periodic)
  04h 3Ch 3Ch  A_UINT8  1  voicePSCapability  ;VOICEPS_CAP_TYPE      (00h)
  05h 3Dh 3Dh  A_UINT8  1  tsid                                      (05h)
  06h 3Eh 3Eh  A_UINT8  1  userPriority       ;802.1D user priority  (00h)
  -   -   3Fh  A_UINT8  1  nominalPHY         ;nominal phy rate
```

Old wmi.h source code from 2006 used 8bit/16bit/32bit parameter ordering, later source code reversed that to 32bit/16bit/8bit and added an extra "nominalPHY" entry. DSi uses the new ordering, but without the extra entry.

DIR_TYPE values:

```
  UPLINK_TRAFFIC         = 0
  DNLINK_TRAFFIC         = 1
  BIDIR_TRAFFIC          = 2
```

TRAFFIC_TYPE values:

```
  TRAFFIC_TYPE_APERIODIC = 0
  TRAFFIC_TYPE_PERIODIC  = 1
```

VOICEPS_CAP_TYPE values:

```
  DISABLE_FOR_THIS_AC    = 0
  ENABLE_FOR_THIS_AC     = 1
  ENABLE_FOR_ALL_AC      = 2
```

XXX see WMI_CRE_PRIORITY_STREAM_REPLY ????

WMIcmd(0006h) - WMI_DELETE_PSTREAM_CMD ;aka WMI_DEL_PRIORITY_STREAM

Parameters (05h bytes):

```
  00h  A_UINT8  1   txQueueNumber
  01h  A_UINT8  1   rxQueueNumber
  02h  A_UINT8  1   trafficDirection
  03h  A_UINT8  1   trafficClass
  04h  A_UINT8  1   tsid
```

XXX see WMI_DEL_PRIORITY_STREAM_REPLY ????

WMIevent(1007h) - WMI_PSTREAM_TIMEOUT_EVENT

Event Data (04h bytes):

```
  00h  A_UINT8  1  txQueueNumber
  01h  A_UINT8  1  rxQueueNumber
  02h  A_UINT8  1  trafficDirection
  03h  A_UINT8  1  trafficClass
```

DSi Atheros Wifi - WMI Roam Functions

WMIcmd(0029h) - WMI_SET_ROAM_CTRL_CMD

Parameters (09h bytes on DSi?, but would be greater if "numBss>1"):

```
      PREPACK union {
  ..    A_UINT8  bssid[ATH_MAC_LEN]       ;WMI_FORCE_ROAM
  ..    A_UINT8  roamMode                 ;WMI_SET_ROAM_MODE
  ..    WMI_BSS_BIAS_INFO bssBiasInfo     ;WMI_SET_HOST_BIAS
  ..    WMI_LOWRSSI_SCAN_PARAMS lrScanParams
      } POSTPACK info
  ..  A_UINT8    roamCtrlType
```

This command is used to influence the Roaming behaviour.
Set the host biases of the BSSs before setting the roam mode as bias based.
WMI_ROAM_CTRL_TYPE, Different types of Roam Control:

```
       WMI_FORCE_ROAM              = 1  ;Roam to the specified BSSID
       WMI_SET_ROAM_MODE           = 2  ;default ,progd bias, no roam
       WMI_SET_HOST_BIAS           = 3  ;Set the Host Bias
       WMI_SET_LOWRSSI_SCAN_PARAMS = 4  ;Set lowrssi Scan parameters
```

WMI_ROAM_MODE, aka ROAM MODES:

```
       WMI_DEFAULT_ROAM_MODE       = 1  ;RSSI based ROAM
       WMI_HOST_BIAS_ROAM_MODE     = 2  ;HOST BIAS based ROAM
       WMI_LOCK_BSS_MODE           = 3  ;Lock to the Current BSS - no Roam
```

BSS HOST BIAS INFO structures:

```
  WMI_BSS_BIAS typedef PREPACK struct:
       6    A_UINT8  bssid[ATH_MAC_LEN]
       1    A_INT8   bias
  WMI_BSS_BIAS_INFO typedef PREPACK struct:
    00h 1    A_UINT8  numBss
    01h 7*N  WMI_BSS_BIAS bssBias[1]
  WMI_LOWRSSI_SCAN_PARAMS typedef PREPACK struct WMI_LOWRSSI_SCAN_PARAMS:
    00h 2    A_UINT16 lowrssi_scan_period
    02h 2    A_INT16  lowrssi_scan_threshold
    04h 2    A_INT16  lowrssi_roam_threshold
    06h 1    A_UINT8  roam_rssi_floor
    07h 1    A_UINT8  reserved[1]              ;for alignment
```

WMIcmd(002Ah) - WMI_GET_ROAM_TBL_CMD ;aka WMI_REPORT_ROAM_TBL ;reply 100Fh
Parameters:

```
  Unknown (none?)
```

Reply: See WMI_REPORT_ROAM_TBL_EVENT

WMIevent(100Fh) - WMI_REPORT_ROAM_TBL_EVENT ;related to CMD 002Ah ?
Event Data (14h bytes on DSi, might be more on other systems if numEntries>1):

```
  00h  A_UINT16           2      roamMode
  02h  A_UINT16           2      numEntries
  04h  WMI_BSS_ROAM_INFO  N*10h  bssRoamInfo[1]
```

WMI_BSS_ROAM_INFO structure:

```
    A_INT32 4  roam_util
    A_UINT8 6  bssid[ATH_MAC_LEN]
    A_INT8  1  rssi
    A_INT8  1  rssidt
    A_INT8  1  last_rssi
    A_INT8  1  util
    A_INT8  1  bias
    A_UINT8 1  reserved         ;for alignment
```

MAX_ROAM_TBL_CAND = 5
Names: WMI_GET_ROAM_TBL aka WMI_REPORT_ROAM_TBL aka WMI_TARGET_ROAM_TBL

DSi Atheros Wifi - WMI Power Functions

WMIcmd(0012h) - WMI_SET_POWER_MODE_CMD ;aka WMI_POWER_MODE_CMD
Parameters (01h bytes):

```
  00h  A_UINT8  1   powerMode           ;WMI_POWER_MODE
```
WMI_POWER_MODE values:
```
  REC_POWER      = 01h     ;recommended,receive,record,rectal?
  MAX_PERF_POWER = 02h     ;maximum performance,perforation?
```

WMIcmd(0013h) - WMI_SET_IBSS_PM_CAPS_CMD ;aka WMI_IBSS_PM_CAPS_CMD
Parameters (06h bytes):
```
  00h  A_UINT8  1  power_saving
  01h  A_UINT8  1  ttl                 ;number of beacon periods
  02h  A_UINT16 2  atim_windows        ;msec
  04h  A_UINT16 2  timeout_value       ;msec
```
Adhoc power save types aka WMI_ADHOC_PS_TYPE:
```
  ADHOC_PS_DISABLE = 1
  ADHOC_PS_ATH     = 2
  ADHOC_PS_IEEE    = 3
  ADHOC_PS_OTHER   = 4
```

WMIcmd(0014h) - WMI_SET_POWER_PARAMS_CMD ;aka WMI_POWER_PARAMS_CMD
Parameters (06h bytes on DSi?, but other sources claim 0Ch bytes):
```
  06h 0Ch   <---- total size (on DSi it's 06h, ie. left column)
  00h 00h  A_UINT16 2   idle_period            ;msec
  02h 02h  A_UINT16 2   pspoll_number
  04h 04h  A_UINT16 2   dtim_policy
  --  06h  A_UINT16 2   tx_wakeup_policy
  --  08h  A_UINT16 2   num_tx_to_wakeup
  --  0Ah  A_UINT16 2   ps_fail_event_policy
```
WMI_DTIM_POLICY values:
```
  IGNORE_DTIM = 01h
  NORMAL_DTIM = 02h
  STICK_DTIM  = 03h
  AUTO_DTIM   = 04h
```
WMI_TX_WAKEUP_POLICY_UPON_SLEEP values (Policy to determnine (Nine?) whether TX should wakeup WLAN if sleeping):
```
  TX_WAKEUP_UPON_SLEEP      = 1
  TX_DONT_WAKEUP_UPON_SLEEP = 2
```
POWER_SAVE_FAIL_EVENT_POLICY values (Policy to determnine (Nine?) whether power save failure event should be sent to host during scanning):
```
  SEND_POWER_SAVE_FAIL_EVENT_ALWAYS        = 1
  IGNORE_POWER_SAVE_FAIL_EVENT_DURING_SCAN = 2
```

WMIcmd(0015h) - WMI_SET_POWERSAVE_TIMERS_POLICY_CMD ;aka WMI_POWERSAVE...
Parameters (0Ch bytes):
```
  00h  A_UINT16 2  psPollTimeout (msec)
  02h  A_UINT16 2  triggerTimeout (msec)
  04h  A_UINT32 4  apsdTimPolicy (TIM behavior with ques (=?) APSD enabled.
                    Default is IGNORE_TIM_ALL_QUEUES_APSD)
  08h  A_UINT32 4  simulatedAPSDTimPolicy (TIM behavior with simulated APSD
                    enabled. Default is PROCESS_TIM_SIMULATED_APSD)
```
APSD_TIM_POLICY values:
```
  IGNORE_TIM_ALL_QUEUES_APSD  = 0
  PROCESS_TIM_ALL_QUEUES_APSD = 1
  IGNORE_TIM_SIMULATED_APSD   = 2
  PROCESS_TIM_SIMULATED_APSD  = 3
```

WMIcmd(001Bh) - WMI_SET_TX_PWR_CMD
Parameters:
```
  00h  A_UINT8  1   dbM                ;in dbM units
```

WMIcmd(001Ch) - WMI_GET_TX_PWR_CMD ;aka WMI_TX_PWR ;reply 001Ch
Parameters:

```
  None
```
Reply:
```
  00h  A_UINT8  1   dbM                ;in dbM units
```

WMIcmd(0047h) - WMI_START_WHATEVER_TIMER_CMD ;-special DSi command?
This seems to be a nintendo-specific command. It seems to be related to power control functions (maybe similar to WMI_SET_AP_PS_CMD, but with changed command number and fewer parameters). The command seems to start a timer, and the firmware perhaps does something when the timer has ellapsed?
Parameters (04h bytes):
```
  -   00h  A_UINT32 4   time           ;in msec or so?   ;used: 00000002h
```

WMIcmd(0048h) - WMI_SET_AP_PS_CMD ;aka WMI_AP_PS_CMD ;not implemented on DSi
This supported in most or all atheros code versions, but DSi doesn't support it.
Parameters (0Ah bytes):
```
  -   00h  A_UINT32 4   idle_time      ;in msec
  -   04h  A_UINT32 4   ps_period      ;in usec
  -   08h  A_UINT8  1   sleep_period   ;in ps_period's (=above "ps_period"?)
  -   09h  A_UINT8  1   psType         ;AP power save type
```
WMI_AP_PS_TYPE, AP power save types:
```
  AP_PS_DISABLE = 1
  AP_PS_ATH     = 2
```

DSi Atheros Wifi - WMI Statistics Function

WMIcmd(0010h) - WMI_GET_STATISTICS_CMD ;reply WMI_REPORT_STATISTICS
Parameters:
```
  Unknown (none?)
```

WMIevent(100Bh) - WMI_REPORT_STATISTICS_EVENT ;related to CMD 0010h
Event Data (D5h bytes on DSi, although other sources claim A9h or EDh bytes):
```
  A9h D5h EDh  <---- total size (on DSi it's D5h, ie. middle column)
  --  00h 00h A_UINT32 4  lqVal                    ;-  <-- newer version only
  --  04h 04h A_INT32  4  noise_floor_calibation   ;-  <-- newer version only
  --  08h 08h A_UINT32 4  power_save_failure_cnt   ;\pm_stats_t (new)
  --  --  0Ch A_UINT16 2  stop_tx_failure_cnt      ;   <-- NEWEST version only
  --  --  0Eh A_UINT16 2  atim_tx_failure_cnt      ;   <-- NEWEST version only
  --  --  10h A_UINT16 2  atim_rx_failure_cnt      ;   <-- NEWEST version only
  --  --  12h A_UINT16 2  bcn_rx_failure_cnt       ;/  <-- NEWEST version only
  00h 0Ch 14h A_UINT32 4  tx_packets               ;\
  04h 10h 18h A_UINT32 4  tx_bytes                 ;
  08h 14h 1Ch A_UINT32 4  tx_unicast_pkts          ;
  0Ch 18h 20h A_UINT32 4  tx_unicast_bytes         ; tx_stats_t
  10h 1Ch 24h A_UINT32 4  tx_multicast_pkts        ;
  14h 20h 28h A_UINT32 4  tx_multicast_bytes       ;
  18h 24h 2Ch A_UINT32 4  tx_broadcast_pkts        ;
  1Ch 28h 30h A_UINT32 4  tx_broadcast_bytes       ;
  20h 2Ch 34h A_UINT32 4  tx_rts_success_cnt       ;
  24h 30h 38h A_UINT32 16 tx_packet_per_ac[4]      ;
  --  40h 48h A_UINT32 16 tx_errors_per_ac[4]      ;   <-- newer version only
  34h 50h 58h A_UINT32 4  tx_errors                ;
  38h 54h 5Ch A_UINT32 4  tx_failed_cnt            ;
  3Ch 58h 60h A_UINT32 4  tx_retry_cnt             ;
  --  --  64h A_UINT32 4  tx_mult_retry_cnt        ;   <-- NEWEST version only
  40h 5Ch 68h A_UINT32 4  tx_rts_fail_cnt          ;
  --  60h 6Ch A_INT32  4  tx_unicast_rate          ;/  <-- newer version only
  44h 64h 70h A_UINT32 4  rx_packets               ;\
  48h 68h 74h A_UINT32 4  rx_bytes                 ;
  4Ch 6Ch 78h A_UINT32 4  rx_unicast_pkts          ;
  50h 70h 7Ch A_UINT32 4  rx_unicast_bytes         ;
```

```
  54h 74h 80h A_UINT32 4  rx_multicast_pkts        ; rx_stats_t
  58h 78h 84h A_UINT32 4  rx_multicast_bytes       ;
  5Ch 7Ch 88h A_UINT32 4  rx_broadcast_pkts        ;
  60h 80h 8Ch A_UINT32 4  rx_broadcast_bytes       ;
  64h 84h 90h A_UINT32 4  rx_fragment_pkt          ;
  68h 88h 94h A_UINT32 4  rx_errors                ;
  6Ch 8Ch 98h A_UINT32 4  rx_crcerr                ;
  70h 90h 9Ch A_UINT32 4  rx_key_cache_miss        ;
  74h 94h A0h A_UINT32 4  rx_decrypt_err           ;
  78h 98h A4h A_UINT32 4  rx_duplicate_frames      ;
  --  9Ch A8h A_INT32  4  rx_unicast_rate          ;/  <-- newer version only
  7Ch A0h ACh A_UINT32 4  tkip_local_mic_failure         ;\
  80h A4h B0h A_UINT32 4  tkip_counter_measures_invoked  ;
  84h A8h B4h A_UINT32 4  tkip_replays                   ; tkip_ccmp_stats_t
  88h ACh B8h A_UINT32 4  tkip_format_errors             ;
  8Ch B0h BCh A_UINT32 4  ccmp_format_errors             ;
  90h B4h C0h A_UINT32 4  ccmp_replays                   ;/
  --  B8h C4h A_UINT32 4  wow_num_pkts_dropped           ;\
  --  BCh C8h A_UINT16 2  wow_num_events_discarded       ; wlan_wow_stats_t
  --  BEh CAh A_UINT8  1  wow_num_host_pkt_wakeups       ;
  --  BFh CBh A_UINT8  1  wow_num_host_event_wakeups     ;/
  --  --  CCh A_UINT32 4  arp_received                   ;\
  --  --  D0h A_UINT32 4  arp_matched                    ; arp_stats_t
  --  --  D4h A_UINT32 4  arp_replied                    ;/
  94h C0h D8h A_UINT32 4  cs_bmiss_cnt            ;\
  98h C4h DCh A_UINT32 4  cs_lowRssi_cnt          ;
  9Ch C8h E0h A_UINT16 2  cs_connect_cnt          ; cserv_stats_t
  9Eh CAh E2h A_UINT16 2  cs_disconnect_cnt       ;
  A0h CCh E4h A_INT16  2  cs_aveBeacon_rssi       ;
  --  CEh E6h A_UINT16 2  cs_roam_count           ;   <-- newer version only
  --  D0h E8h A_INT16  2  cs_rssi                 ;   <-- newer version only
  --  D2h EAh A_UINT8  1  cs_snr                  ;   <-- newer version only
  --  D3h EBh A_UINT8  1  cs_aveBeacon_snr        ;   <-- newer version only
  A2h D4h ECh A_UINT8  1  cs_lastRoam_msec        ;/
  A3h --  --  A_UINT32 4  power_save_failure_cnt  ;-pm_stats_t (old)
  A7h --  --  A_INT16  2  noise_floor_calibation  ;-old version only
```

The D5h-byte structure was found in AR6kSDK.build_sw.18 from 2006.
The A9h-byte structure was found in older (AR6001?) source from 2006.
The EDh-byte structure was found in newer source from 2008 and up.
Names: WMI_GET_STATISTICS aka WMI_REPORT_STATISTICS aka WMI_TARGET_STATS

DSi Atheros Wifi - WMI Bluetooth Coexistence (older AR6002)

Bluetooth Coexistence support has underwent significant changes:

```
  Older AR6002 from 2008 ;-original Bluetooth COEX version
  Newer AR6002 from 2008 ;\same commands as above, but with entirely different
  Newer AR6002 from 2010 ;/parameters (and minor changes for 2008 vs 2010)
  AR6003 from 2010       ;-completely different commands for Bluetooth COEX
```

Below are Bluetooth COEX functions for OLDER AR6002 - as used on DSi - and as defined in "AR6kSDK.build_sw.18".

On DSi, AR6002 does handle command 003Ch and 003Ch, but AR6013/AR6014 do merely redirect them to dummy handlers; despite of also having (unused) actual handlers for those commands.

WMIcmd(003Bh) - WMI_SET_BT_STATUS_CMD ;AR6002 Bluetooth Coexistence only?
Parameters (02h bytes):

```
  00h  A_UINT8  1   streamType          ;aka BT_STREAM_TYPE ?
  01h  A_UINT8  1   status              ;aka BT_STREAM_STATUS ?
```

BT_STREAM_TYPE values:

```
  BT_STREAM_UNDEF = 0                   ;\
```

```
  BT_STREAM_SCO   = 1 ;SCO stream       ; only this three types in Older AR6002
  BT_STREAM_A2DP  = 2 ;A2DP stream      ;/
```
BT_STREAM_STATUS values:
```
  BT_STATUS_UNDEF        = 0  ;\
  BT_STATUS_START        = 1  ; this five states in Older AR6002
  BT_STATUS_STOP         = 2  ; (Newer A6002 has added/removed/renamed states)
  BT_STATUS_RESUME       = 3  ;
  BT_STATUS_SUSPEND      = 4  ;/
```
AR6002 only? (although other comment says "COMMON to AR6002 and AR6003"?)

WMIcmd(003Ch) - WMI_SET_BT_PARAMS_CMD ;AR6002 Bluetooth Coexistence only?
Parameters (16h bytes on DSi?, whatever that might match up with below?):
```
 when paramType=1=BT_PARAM_SCO:    ;SCO stream parameters (BT_PARAMS_SCO)
  00h  A_UINT8  1   noSCOPkts
  01h  A_UINT8  1   pspollTimeout
  02h  A_UINT8  1   stompbt
  03h  PAD      12h undefined/padding
 when paramType=2=BT_PARAM_A2DP:   ;whatever (BT_PARAMS_A2DP)
  00h  A_UINT32 4   period
  04h  A_UINT32 4   dutycycle
  08h  A_UINT8  1   stompbt
  09h  PAD      0Ch undefined/padding
 when paramType=3=BT_PARAM_MISC and paramSubType=1=WLAN_PROTECT_POLICY:
  00h  A_UINT32 4   period
  04h  A_UINT32 4   dutycycle
  08h  A_UINT8  1   stompbt
  09h  A_UINT8  1   policy
  0Ah  A_UINT8  1   paramSubType  (=1 in this case)
  0Bh  PAD      0Ah undefined/padding
 when paramType=3=BT_PARAM_MISC and paramSubType=2=WLAN_COEX_CTRL_FLAGS:
  00h  A_UINT16 2   wlanCtrlFlags
  02h  PAD      8   undefined/padding
  0Ah  A_UINT8  1   paramSubType  (=2 in this case)
  0Bh  PAD      0Ah undefined/padding
 when paramType=4=BT_PARAM_REGS:   ;co-existence register params (BT_COEX_REGS)
  00h  A_UINT32 4   mode
  04h  A_UINT32 4   scoWghts
  08h  A_UINT32 4   a2dpWghts
  0Ch  A_UINT32 4   genWghts
  10h  A_UINT32 4   mode2
  14h  A_UINT8  1   setVal
 and, in all cases:
  15h  A_UINT8  1   paramType    ;<-- selects which of the above to use
```

Below might be "policy" for WLAN_PROTECT_POLICY(?):
```
  WLAN_PROTECT_PER_STREAM = 01h   /* default
  WLAN_PROTECT_ANY_TX     = 02h
```

Below might be "wlanCtrlFlags" for WLAN_COEX_CTRL_FLAGS(?):
```
  WLAN_DISABLE_COEX_IN_DISCONNECT = 0001h   ;default
  WLAN_KEEP_COEX_IN_DISCONNECT    = 0002h
  WLAN_STOMPBT_IN_DISCONNECT      = 0004h
  WLAN_DISABLE_COEX_IN_ROAM       = 0010h   ;default
  WLAN_KEEP_COEX_IN_ROAM          = 0020h
  WLAN_STOMPBT_IN_ROAM            = 0040h
  WLAN_DISABLE_COEX_IN_SCAN       = 0100h   ;default
  WLAN_KEEP_COEX_IN_SCAN          = 0200h
  WLAN_STOMPBT_IN_SCAN            = 0400h
  WLAN_DISABLE_COEX_BT_OFF        = 1000h   ;default
  WLAN_KEEP_COEX_BT_OFF           = 2000h
  WLAN_STOMPBT_BT_OFF             = 4000h
```

DSi Atheros Wifi - WMI Wake on Wireless (WOW) Functions

WMIcmd(0042h) - WMI_SET_HOST_SLEEP_MODE_CMD
Parameters (08h bytes on DSi?, so, a BOOL must be 4 bytes?):
```
  00h  A_BOOL   4  awake;
  04h  A_BOOL   4  asleep;
```
See also: WMI_SET_HOST_SLEEP_MODE_CMD_PROCESSED_EVENT

WMIcmd(0043h) - WMI_SET_WOW_MODE_CMD
Event Data (04h bytes on DSi?, but other sources claim MORE bytes?):
```
  04h ??h   <---- total size (on DSi it's 04h, ie. left column)
  00h 00h  A_BOOL         4   enable_wow
  --  04h  WMI_WOW_FILTER ..  filter ;UINTx or so? with "WMI_WOW_FILTER" value?
  --  ..   A_UINT16       2   hostReqDelay
```
WMI_WOW_FILTER values (only one defined):
```
  WOW_FILTER_SSID = 01h
```

WMIcmd(0044h) - WMI_GET_WOW_LIST_CMD ;reply 1018h (!)
Parameters:
```
  00h  A_UINT8  1   filter_list_id;
```
Reply (88h bytes on DSi): WMI_GET_WOW_LIST_EVENT:
```
  00h  A_UINT8    1      num_filters     /* number of patterns in reply
  01h  A_UINT8    1      this_filter_num /* filter # x of total num_filters
  02h  A_UINT8    1      wow_mode
  03h  A_UINT8    1      host_mode
  04h  WOW_FILTER N*84h  wow_filters[1]
```
WOW_FILTER structure:
```
    A_UINT8 1    wow_valid_filter;
    A_UINT8 1    wow_filter_id;
    A_UINT8 1    wow_filter_size;
    A_UINT8 1    wow_filter_offset;
    A_UINT8 40h  wow_filter_mask[WOW_MASK_SIZE];
    A_UINT8 40h  wow_filter_pattern[WOW_PATTERN_SIZE];
```
There's also a "WOW_FILTER_LIST" structure (unknown purpose):
```
    A_UINT8    1      wow_valid_list;
    A_UINT8    1      wow_list_id;
    A_UINT8    1      wow_num_filters;
    A_UINT8    1      wow_total_list_size;
    WOW_FILTER 4*84h  list[WOW_MAX_FILTERS_PER_LIST];
 #define WOW_MAX_FILTER_LISTS     = 1     /*  4  */
 #define WOW_MAX_FILTERS_PER_LIST = 4
 #define WOW_PATTERN_SIZE         = 64
 #define WOW_MASK_SIZE            = 64
```

WMIcmd(0045h) - WMI_ADD_WOW_PATTERN_CMD
Parameters:
```
  00h  A_UINT8  1   filter_list_id;
  01h  A_UINT8  1   filter_size;
  02h  A_UINT8  1   filter_offset;
  03h  A_UINT8  ..  filter[1];
```

WMIcmd(0046h) - WMI_DEL_WOW_PATTERN_CMD
Parameters (04h bytes):
```
  00h  A_UINT16 2   filter_list_id;
  02h  A_UINT16 2   filter_id;
```

WMIevent(1018h) - WMI_GET_WOW_LIST_EVENT ;reply to CMD 0044h
See WMI_GET_WOW_LIST_CMD for response details.

DSi Atheros Wifi - WMI General Purpose I/O (GPIO) Functions

WMIcmd(002Eh:2003h) - WMIX_GPIO_OUTPUT_SET_CMD ;reply=3006h

Parameters:

```
  00h  A_UINT32  4     set_mask;             /* pins to set
  04h  A_UINT32  4     clear_mask;           /* pins to clear
  08h  A_UINT32  4     enable_mask;          /* pins to enable for output
  0Ch  A_UINT32  4     disable_mask;         /* pins to disable/tristate
```

Set GPIO pin output state.
In order for output to be driven, a pin must be enabled for output.
This can be done during initialization through the GPIO Configuration DataSet, or during operation with the enable_mask.
If a request is made to simultaneously set/clear or set/disable or clear/disable or disable/enable, results are undefined.
NB: Some of the WMIX APIs use a 32-bit mask. On Targets that support more than 32 GPIO pins, those APIs only support the first 32 GPIO pins.

WMIcmd(002Eh:2004h) - WMIX_GPIO_INPUT_GET_CMD ;reply=3005h

Parameters:

```
  Unknown (none?)
```

WMIcmd(002Eh:2005h) - WMIX_GPIO_REGISTER_SET_CMD ;reply=3006h, too

Parameters:

```
  00h  A_UINT32  4    gpioreg_id;           /* GPIO register ID
  04h  A_UINT32  4    value;                /* value to write
```

Set a GPIO register. For debug/exceptional cases.
Values for gpioreg_id are GPIO_ID_*, defined in a platform-dependent header, gpio.h.

WMIcmd(002Eh:2006h) - WMIX_GPIO_REGISTER_GET_CMD ;reply=3005h, too

Parameters:

```
  00h  A_UINT32  4    gpioreg_id;           /* GPIO register to read
```

Get a GPIO register. For debug/exceptional cases.

WMIcmd(002Eh:2007h) - WMIX_GPIO_INTR_ACK_CMD

Parameters:

```
    A_UINT32              ack_mask;             /* interrupts to acknowledge
```

Host acknowledges and re-arms GPIO interrupts. A single message should be used to acknowledge all interrupts that were delivered in an earlier WMIX_GPIO_INTR_EVENT message.

WMIevent(1010h:3004h) - WMIX_GPIO_INTR_EVENT ;used (interrupt)

Event Data:

```
  00h  A_UINT32  4   intr_mask;            /* pending GPIO interrupts
  04h  A_UINT32  4   input_values;         /* recent GPIO input values
```

Target informs Host of GPIO interrupts that have ocurred since the last WMIX_GIPO_INTR_ACK_CMD was received. Additional information -- the current GPIO input values is provided -- in order to support use of a GPIO interrupt as a Data Valid signal for other GPIO pins.

WMIevent(1010h:3005h) - WMIX_GPIO_DATA_EVENT ;used (reply to 2004h and 2006h)

Event Data:

```
  00h  A_UINT32  4   value;
  04h  A_UINT32  4   reg_id;
```

Target responds to Host's earlier WMIX_GPIO_INPUT_GET_CMD request using a GPIO_DATA_EVENT with value set to the mask of GPIO pin inputs and reg_id set to GPIO_ID_NONE.
Target responds to Hosts's earlier WMIX_GPIO_REGISTER_GET_CMD request using a

GPIO_DATA_EVENT with value set to the value of the requested register and reg_id identifying the register (reflects the original request).
NB: reg_id supports the future possibility of unsolicited WMIX_GPIO_DATA_EVENTs (for polling GPIO input), and it may simplify Host GPIO support.

WMIevent(1010h:3006h) - WMIX_GPIO_ACK_EVENT ;used (reply to 2003h and 2005h)
Event Data:
```
  Unknown (none?) (confirms GPIO_xxx_SET commands)
```

GPIO Constants
```
  AR6001_GPIO_PIN_COUNT   = 18
  AR6002_GPIO_PIN_COUNT   = 18  ;aka hw2.0
  AR6003_GPIO_PIN_COUNT   = 28  ;aka hw4.0 ;XXX shouldn't that be 26 ?
  MCKINLEY_GPIO_PIN_COUNT = 57  ;aka hw6.0
```
Values of gpioreg_id in the WMIX_GPIO_REGISTER_SET_CMDID and WMIX_GPIO_REGISTER_GET_CMDID commands come in two flavors. If the upper bit of gpioreg_id is CLEAR, then the remainder is interpreted as one of these values. This provides platform-independent access to GPIO registers. If the upper bit (GPIO_ID_OFFSET_FLAG) of gpioreg_id is SET, then the remainder is interpreted as a platform-specific GPIO register offset.
```
  GPIO_ID_OUT         = 00000000h
  GPIO_ID_OUT_W1TS    = 00000001h
  GPIO_ID_OUT_W1TC    = 00000002h
  GPIO_ID_ENABLE      = 00000003h
  GPIO_ID_ENABLE_W1TS = 00000004h
  GPIO_ID_ENABLE_W1TC = 00000005h
  GPIO_ID_IN          = 00000006h
  GPIO_ID_STATUS      = 00000007h
  GPIO_ID_STATUS_W1TS = 00000008h
  GPIO_ID_STATUS_W1TC = 00000009h
  GPIO_ID_PIN0        = 0000000Ah
  GPIO_ID_PIN(n)      = (GPIO_ID_PIN0+(n))  ;=0000000Ah and up
  GPIO_ID_NONE        = FFFFFFFFh
  GPIO_ID_OFFSET_FLAG       = 80000000h
  GPIO_ID_REG_MASK          = 7fffffffh
  GPIO_ID_IS_OFFSET(reg_id) = (((reg_id) & GPIO_ID_OFFSET_FLAG) != 0)
```

DSi Atheros Wifi - Unimplemented WMI Misc Functions

Not implemented in DSi.

WMIcmd(002Bh) - WMI_GET_ROAM_DATA_CMD ;reply 1015h ? ;not implemented in DSi
Parameters:
```
  Unknown (none?)
```
Reply: See WMI_REPORT_ROAM_DATA_EVENT

WMIevent(1015h) - WMI_REPORT_ROAM_DATA_EVENT
```
  ;-not implemented in DSi ;related to 002Bh?
```
Event Data:
```
       PREPACK union {
  00h      WMI_TARGET_ROAM_TIME roamTime;
       } POSTPACK u;
  14h  A_UINT8 roamDataType ;
```
ROAM_DATA_TYPE values (only one defined)
```
  ROAM_DATA_TIME = 1          /* Get The Roam Time Data
```
WMI_TARGET_ROAM_TIME structure:
```
  00h  A_UINT32  4     disassoc_time
  04h  A_UINT32  4     no_txrx_time
  08h  A_UINT32  4     assoc_time
```

```
  0Ch  A_UINT32  4     allow_txrx_time
  10h  A_UINT8   6     disassoc_bssid[ATH_MAC_LEN]
  16h  A_INT8    1     disassoc_bss_rssi
  17h  A_UINT8   6     assoc_bssid[ATH_MAC_LEN]      ;UNALIGNED!!!
  1Dh  A_INT8    1     assoc_bss_rssi
```
Names: WMI_GET_ROAM_DATA aka WMI_REPORT_ROAM_DATA aka WMI_TARGET_ROAM_DATA

WMIcmd(002Ch) - WMI_ENABLE_RM_CMD ;not implemented in DSi
Parameters:
```
  00h  A_BOOL   4    enable_radio_measurements;
```

WMIcmd(002Dh) - WMI_SET_MAX_OFFHOME_DURATION_CMD ;not implemented in DSi
Parameters:
```
  00h  A_UINT8  1    max_offhome_duration;
```

WMIcmd(002Eh:200Ah) - WMIX_PROF_CFG_CMD
WMIcmd(002Eh:200Bh) - WMIX_PROF_ADDR_SET_CMD
WMIcmd(002Eh:200Ch) - WMIX_PROF_START_CMD
WMIcmd(002Eh:200Dh) - WMIX_PROF_STOP_CMD
WMIcmd(002Eh:200Eh) - WMIX_PROF_COUNT_GET_CMD ;reply 3009h
Not implemented in DSi. Said to be "Target Profiling support".
Parameter structures are defined only for WMIX_PROF_CFG_CMD and WMIX_PROF_ADDR_SET_CMD.
Parameters for WMIX_PROF_CFG_CMD:
```
  00h  A_UINT32 4   period;     /* Time (in 30.5us ticks) between samples
  04h  A_UINT32 4   nbins;
```
Parameters for WMIX_PROF_ADDR_SET_CMD:
```
  00h  A_UINT32 4   addr;
```
Maybe the other three WMIX_PROF_xxx_CMD's don't have any parameters.
See also: WMIX_PROF_COUNT_EVENT

WMIevent(1010h:3009h) - WMIX_PROF_COUNT_EVENT ;-not implemented in DSi
Not implemented in DSi. Response to WMIX_PROF_COUNT_GET_CMD.
Event Data:
```
  00h  A_UINT32  4     addr;
  04h  A_UINT32  4     count;
```
Target responds to Hosts's earlier WMIX_PROF_COUNT_GET_CMD request using a WMIX_PROF_COUNT_EVENT with addr set to the next address count set to the corresponding count.

WMIcmd(003Ah) - WMI_TEST_CMD ;not implemented in DSi
Parameters:
```
  Unknown (maybe related to file "testcmd.h"?)
```

WMIevent(1016h) - WMI_TEST_EVENT ;not implemented in DSi
Event Data:
```
  Unknown (maybe related to file "testcmd.h"?) (or general purpose?)
```

WMIcmd(0040h) - WMI_GET_APPIE_CMD ;aka GET_APP_IE ;not implemented in DSi
Parameters:
```
  Unknown (none?)
```
Reply: Unknown:
```
  EVENTID is unknown (maybe 0040h, ie. same as GET_APPIE_CMD)
  Reply structure is unknown (maybe same parameter structure for SET_APPIE_CMD)
```

WMIcmd(004Ah) - WMI_SET_IE_CMD ;not implemented in DSi (newer 2012 stuff)
Parameters:
```
  00h   u8  1   ie_id;
  01h   u8  1   ie_field;    /* enum wmi_ie_field_type
```

```
  02h   u8  1   ie_len;
  03h   u8  1   reserved;
  04h   u8  ..  ie_info[0];
```
wmi_ie_field_type:
```
  WMI_RSN_IE_CAPB = 01h
  WMI_IE_FULL     = FFh  /* indicats full IE    ;uh, kittykats?
```
See also: WMI_SET_APP_IE_CMD (similar older command)

WMIcmd(08xxh) - wil6210: WILOCITY types ;not implemented in DSi
WMIcmd(09xxh) - wil6210: Performance monitoring ;not implemented in DSi
```
  not implemented in DSi
```

WMIcmd(F003h, or formerly 0047h) - WMI_SET_MAC_ADDRESS_CMD
Parameters:
```
  00h  A_UINT8  6   macaddr[ATH_MAC_LEN];
```

WMIcmd(F007h) - WMI_ABORT_SCAN_CMD ;not implemented in DSi
Parameters:
```
  Unknown (none?)
```
Reply: Unknown, if any (see (optional?) SCAN_ABORT_EVENT)

WMIcmd(F008h) - WMI_SET_TARGET_EVENT_REPORT_CMD ;not implemented in DSi
Parameters:
```
  00h  A_UINT32 1   evtConfig;
```
TARGET_EVENT_REPORT_CONFIG values:
```
  DISCONN_EVT_IN_RECONN    = 0   ;default
  NO_DISCONN_EVT_IN_RECONN = 1
```
Apparently related to cases where to throw WMI_DISCONNECT_EVENT.

WMIcmd(F017h or formerly F016h) - WMI_SET_IP_CMD
Parameters:
```
  00h  A_UINT32 4*2  ips[MAX_IP_ADDRS]   ;IP in Network Byte Order
```
#define MAX_IP_ADDRS 2

WMIcmd(F018h or formerly F017h) - WMI_SET_PARAMS_CMD ;reply=101Fh
Parameters:
```
  00h  A_UINT32 4   opcode;
  04h  A_UINT32 4   length;
  08h  A_CHAR   ... buffer[1];      /* WMI_SET_PARAMS
```
Reply: See WMI_SET_PARAMS_REPLY_EVENT

WMIevent(101Fh) - WMI_SET_PARAMS_REPLY_EVENT ;reply to "SET" CMD F018h
Event Data:
```
  00h  A_INT8  1   status;      /* WMI_SET_PARAMS_REPLY
```
Reply to WMI_SET_PARAMS_CMD (?) aka WMI_SET_PARAMS_REPLY aka WMI_SET_PARAMS_REPLY_EVENT.

WMIcmd(F019h or formerly F018h) - WMI_SET_MCAST_FILTER_CMD
Parameters:
```
  00h  A_UINT8  6   multicast_mac[ATH_MAC_LEN];   /* WMI_SET_MCAST_FILTER
```

WMIcmd(F01Ah or formerly F019h) - WMI_DEL_MCAST_FILTER_CMD
Parameters:
```
  Unknown (None?) (or maybe same as for WMI_SET_MCAST_FILTER_CMD ?)
```

WMIcmd(F029h) - WMI_MCAST_FILTER_CMD ;related to SET/DEL "MCAST" commands?
Parameters:

```
  00h  A_UINT8  1  enable;      /* WMI_MCAST_FILTER
```
Related to SET/DEL "MCAST" commands?

WMIcmd(F01Bh) - WMI_ALLOW_AGGR_CMD
Parameters:
```
  00h  A_UINT16 2   tx_allow_aggr (16bit mask to allow tx/uplink ADDBA
                     negotiation - bit position indicates tid)
  02h  A_UINT16 2   rx_allow_aggr (16bit mask to allow rx/donwlink ADDBA
                     negotiation - bit position indicates tid)
```
Configures tid's to allow ADDBA negotiations on each tid, in each direction.
uh, donwlink?

WMIcmd(F01Ch) - WMI_ADDBA_REQ_CMD
Parameters:
```
  00h  A_UINT8  1   tid
```
"f/w starts performing ADDBA negotiations with peer on the given tid"
"f/w" means FirmWare? ForWard? Fail/Wrong? or What?

WMIcmd(F01Dh) - WMI_DELBA_REQ_CMD
Parameters:
```
  00h  A_UINT8  1   tid
  01h  A_UINT8  1   is_sender_initiator
```
"f/w would teardown BA with peer." - uh, "f/w"?
"is_send_initiator indicates if it's or tx or rx side" - uh, "it's or"?

WMIevent(1020h) - WMI_ADDBA_REQ_EVENT
Event Data:
```
  00h  A_UINT8  1   tid
  01h  A_UINT8  1   win_sz
  02h  A_UINT16 2   st_seq_no
  04h  A_UINT8  1   status  "f/w response for ADDBA Req; OK(0) or failure(!=0)"
```

WMIevent(1021h) - WMI_ADDBA_RESP_EVENT
Event Data:
```
  00h  A_UINT8  1   tid
  01h  A_UINT8  1   status      /* OK(0), failure (!=0)
  02h  A_UINT16 2   amsdu_sz    /* Three values: Not supported(0), 3839, 8k
```
Uhm, does "8k" mean 8192 or 8000 or so?

WMIevent(1022h) - WMI_DELBA_REQ_EVENT aka WMI_DELBA_EVENT
Event Data:
```
  00h  A_UINT8  1   tid;
  01h  A_UINT8  1   is_peer_initiator;
  02h  A_UINT16 2   reason_code;
```
"f/w received a DELBA for peer and processed it. Host is notified of this."

WMIcmd(F01Eh) - WMI_SET_HT_CAP_CMD
Parameters:
```
  00h  A_UINT8 1  band (specifies which band to apply these values)
  01h  A_UINT8 1  enable (allows 11n to be disabled on a per band basis)
  02h  A_UINT8 1  chan_width_40M_supported
  03h  A_UINT8 1  short_GI_20MHz
  04h  A_UINT8 1  short_GI_40MHz
  05h  A_UINT8 1  intolerance_40MHz
  06h  A_UINT8 1  max_ampdu_len_exp
```

WMIcmd(F01Fh) - WMI_SET_HT_OP_CMD
Parameters:

```
  00h  A_UINT8  1   sta_chan_width;
```

WMIcmd(F020h) - WMI_SET_TX_SELECT_RATES_CMD
Parameters:
```
  00h  A_UINT32 4*8*2  rateMasks[WMI_MODE_MAX * WMI_MAX_RATE_MASK];
```

WMIcmd(F021h) - WMI_SET_TX_SGI_PARAM_CMD
Parameters:
```
  00h  A_UINT32 4*2  sgiMask[WMI_MAX_RATE_MASK];
  08h  A_UINT8  1    sgiPERThreshold;
```
DEFAULT_SGI_MASK_L32 = 08080000h
DEFAULT_SGI_MASK_U32 = 00000000h
DEFAULT_SGI_PER = 10

WMIcmd(F022h) - WMI_SET_RATE_POLICY_CMD
Parameters:
```
  00h  A_UINT32 4*2  rateField[WMI_MAX_RATE_MASK]
                      (rateField: "1 bit per rate corresponding to index")
  08h  A_UINT8  1    id    ;range 1..5 (aka 1..WMI_RATE_POLICY_ID_MAX)
  09h  A_UINT8  1    shortTrys
  0Ah  A_UINT8  1    longTrys
  0Bh  A_UINT8  1    reserved   ;padding
```
WMI_RATE_POLICY_ID_MAX = 5

WMIcmd(F023h) - WMI_HCI_CMD_CMD aka WMI_HCI_CMD
Parameters:
```
  00h  A_UINT16 2   cmd_buf_sz      ;HCI cmd buffer size
  02h  A_UINT8  ..  buf[1]          ;Absolute HCI cmd (see file "hci.h")
```

WMIevent(1024h) - WMI_HCI_EVENT_EVENT aka WMI_HCI_EVENT
Event Data:
```
  00h  A_UINT16 2   evt_buf_sz      ;HCI event buffer size
  02h  A_UINT8  ..  buf[1]          ;HCI event (see file "hci.h")
```

WMIcmd(F024h) - WMI_RX_FRAME_FORMAT_CMD
Parameters:
```
  00h  A_UINT8  1  metaVersion  ;version of meta data for rx packets
                               ;(0-7=valid, 0=default)
  01h  A_UINT8  1  dot11Hdr  ;1=leave .11 header intact,
                             ;0=default/replace .11 header with .3
  02h  A_UINT8  1  defragOnHost  ;1=defragmentation is performed by host,
                                 ;0=performed by target <default>
  03h  A_UINT8  1  reserved[1]     ;alignment
```

WMIcmd(F025h) - WMI_SET_THIN_MODE_CMD
Parameters:
```
  00h  A_UINT8  1  enable      ;0=default/normal mode, 1=operate in thin mode
  01h  A_UINT8  3  reserved[3]
```

WMIcmd(F026h) - WMI_SET_BT_WLAN_CONN_PRECEDENCE_CMD
Parameters:
```
  00h  A_UINT8  1  precedence;
```
BT_WLAN_CONN_PRECEDENCE values:
```
  BT_WLAN_CONN_PRECDENCE_WLAN = 0  ;default
  BT_WLAN_CONN_PRECDENCE_PAL  = 1
```
Unknown purpose. Maybe related to BT=Bluetooth? CONN=Connect? PAL=What?

WMIcmd(F03Fh) - WMI_CONFIG_TX_MAC_RULES_CMD
Parameters:
```
  00h  A_UINT32 4  rules  ;combination of WMI_WRT_xxx values (see "wmi_thin.h")
```

WMIcmd(F040h) - WMI_SET_PROMISCUOUS_MODE_CMD
Parameters:
```
  00h  A_UINT8  1  enable (0=default/normal mode, 1=promiscuous mode)
```

WMIcmd(F041h) - WMI_RX_FRAME_FILTER_CMD
Parameters:
```
  00h  A_UINT16 2  filtermask(0)      ;WMI_FILTERMASK_MGMT
  02h  A_UINT16 2  filtermask(1)      ;WMI_FILTERMASK_CTRL
  04h  A_UINT16 2  filtermask(2)      ;WMI_FILTERMASK_DATA
  06h  A_UINT16 2  reserved           ;alignment
```

WMIcmd(F042h) - WMI_SET_CHANNEL_CMD
Parameters:
```
  00h  A_UINT16 2  channel   ;frequency in MHz
  --  //A_UINT8 -  mode      ;outcommented (HT20 or HT40 flag?)
  --  //A_UINT8 -  secondary ;outcommented (HT40 2nd channel above/below flag?)
```
See also: WMI_SET_CHANNEL_EVENT

WMIevent(9000h) - WMI_SET_CHANNEL_EVENT
Event Data:
```
  00h  A_UINT8 1  result       ;WMI_SET_CHANNEL_RES (or WMI_THIN_JOIN_RESULT??)
  01h  A_UINT8 3  reserved[3]  ;alignment
```
WMI_SET_CHANNEL_RES values:
```
  WMI_SET_CHANNEL_RES_SUCCESS = 0
  WMI_SET_CHANNEL_RES_FAIL    = 1
```
This is probably the reply to WMI_SET_CHANNEL_CMD (although official comments claim it to be WMI_THIN_JOIN related; probably because of copying/pasting the WMI_THIN_JOIN parameter structure without adjusting the comments).

WMIcmd(F046h) - WMI_SET_DIV_PARAMS_CMD aka WMI_DIV_PARAMS_CMD
Parameters:
```
  00h  A_UINT32 4  divIdleTime;
  04h  A_UINT8  1  antRssiThresh;
  05h  A_UINT8  1  divEnable;
  06h  A_UINT16 2  active_treshold_rate;
```

WMIcmd(F028h) - WMI_SET_PMK_CMD
Parameters:
```
  00h  A_UINT8  20h  pmk[WMI_PMK_LEN];
```
WMI_PMK_LEN = 32

WMIcmd(F047h) - WMI_GET_PMK_CMD ;reply?
Parameters:
```
  Unknown (none?)
```
Reply: See WMI_GET_PMK_EVENT aka WMI_GET_PMK_REPLY

WMIevent(102Ah) - WMI_GET_PMK_EVENT aka WMI_GET_PMK_REPLY
Event Data:
```
  00h  A_UINT8  20h  pmk[WMI_PMK_LEN];
```

WMIcmd(F048h) - WMI_SET_PASSPHRASE_CMD
Parameters:
```
  00h  A_UCHAR  20h  ssid[WMI_MAX_SSID_LEN];
  20h  A_UINT8  40h  passphrase[WMI_PASSPHRASE_LEN];
```

```
  60h  A_UINT8  1    ssid_len;
  61h  A_UINT8  1    passphrase_len;
```
WMI_PASSPHRASE_LEN = 64

WMIcmd(F049h) - WMI_SEND_ASSOC_RES_CMD ;aka WMI_SEND_ASSOCRES_CMD
Parameters:
```
  00h  A_UINT8 1  host_accept;
  01h  A_UINT8 1  host_reasonCode;
  02h  A_UINT8 1  target_status;
  03h  A_UINT8 6  sta_mac_addr[ATH_MAC_LEN];
  09h  A_UINT8 1  rspType;
```

WMIcmd(F04Ah) - WMI_SET_ASSOC_REQ_RELAY_CMD ;aka WMI_SET_ASSOCREQ_RELAY
Parameters:
```
  00h  A_UINT8 1  enable;
```

WMIevent(9001h) - WMI_ASSOC_REQ_EVENT aka WMI_ASSOCREQ_EVENT
Event Data:
```
  00h  A_UINT8 1  status;
  01h  A_UINT8 1  rspType;
```

WMIcmd(F04Bh or F04Dh) - WMI_ACS_CTRL_CMD ;aka WMI_ACS_CTRL_MSG
Parameters:
```
  00h  A_UINT8 1   ctrl_id;     /* control identifier (aka sub-command?)
  01h  A_UINT8 1   length;      /* number of bytes of data to follow
  02h  A_UINT8 ..  data[1];     /* start of control data
```
WMI_ACS_CTRL_HDR_LEN = (sizeof(WMI_ACS_CTRL_MSG) - sizeof(A_UINT8))

WMIevent(9002h) - WMI_ACS_EVENT ;generic ACS event
Event Data:
```
  00h  A_UINT8 1   event_id;    /* event identifier
  01h  A_UINT8 1   length;      /* number of bytes of data that follows
  02h  A_UINT8 ..  data[1];     /* start of event data
```
WMI_ACS_EVENT_HDR_LEN = (sizeof(WMI_ACS_EVENT_MSG) - sizeof(A_UINT8))

WMIcmd(F04Ch or F052h) - WMI_SET_EXCESS_TX_RETRY_THRES_CMD
Parameters:
```
  00h  A_UINT32 4  threshold;
```

WMIcmd(F061h or F051h) - WMI_FORCE_TARGET_ASSERT_CMD
Parameters:
```
  Unknown (None?)
```

WMIcmd(F04Dh or N/A) - WMI_SET_TBD_TIME_CMD ;added for wmiconfig cmd for TBD
WMIcmd(F04Eh or N/A) - WMI_PKTLOG_ENABLE_CMD
WMIcmd(F04Fh or N/A) - WMI_PKTLOG_DISABLE_CMD
WMIcmd(F062h or N/A) - WMI_SET_PROBED_SSID_EX_CMD
WMIcmd(F063h or N/A) - WMI_SET_NETWORK_LIST_OFFLOAD_CMD
WMIcmd(F064h or N/A) - WMI_SET_ARP_NS_OFFLOAD_CMD
WMIcmd(F065h or N/A) - WMI_ADD_WOW_EXT_PATTERN_CMD
WMIcmd(F066h or N/A) - WMI_GTK_OFFLOAD_OP_CMD
WMIcmd(F067h or N/A) - WMI_REMAIN_ON_CHNL_CMD
WMIcmd(F068h or N/A) - WMI_CANCEL_REMAIN_ON_CHNL_CMD
WMIcmd(F069h or N/A) - WMI_SEND_ACTION_CMD
WMIcmd(F06Ah or N/A) - WMI_PROBE_REQ_REPORT_CMD
WMIcmd(F06Bh or N/A) - WMI_DISABLE_11B_RATES_CMD
WMIcmd(F06Ch or N/A) - WMI_SEND_PROBE_RESPONSE_CMD

Unknown/undocumented (invented 2012 or so).

WMIevent(?) - WMI_GET_APPIE_CMD ;aka GET_APP_IE ;-not implemented in DSi
The "GET_APP" command name suggests that there should be some reply, but:
```
EVENTID is unknown (maybe 0040h, ie. same as GET_APPIE_CMD)
Reply structure is unknown (maybe same parameter structure for SET_APPIE_CMD)
```

WMIevent(?) - WMI_CRE_PRIORITY_STREAM_REPLY ;-not implemented in DSi
Unknown crap. The DSi doesn't send a reply to WMI_CREATE_PSTREAM_CMD, nonetheless, existing source code does have a WMI_CRE_PRIORITY_STREAM_REPLY structure for whatever reason, maybe it's send only in certain firmware version(s), with whatever/unknown WMIevent(xxxx) number. The reply structure is:
```
00h  A_UINT8  1  status;                /* PSTREAM_REPLY_STATUS
01h  A_UINT8  1  txQueueNumber;
02h  A_UINT8  1  rxQueueNumber;
03h  A_UINT8  1  trafficClass;
04h  A_UINT8  1  trafficDirection;      /* DIR_TYPE
```
PSTREAM_REPLY_STATUS values:
```
A_SUCCEEDED = A_OK                   = 0
A_FAILED_DELETE_STREAM_DOESNOT_EXIST = 250
A_SUCCEEDED_MODIFY_STREAM            = 251
A_FAILED_INVALID_STREAM              = 252
A_FAILED_MAX_THINSTREAMS             = 253
A_FAILED_CREATE_REMOVE_PSTREAM_FIRST = 254
```

WMIevent(?) - WMI_DEL_PRIORITY_STREAM_REPLY ;-not implemented in DSi
Unknown crap. See above for details. The WMI_DEL_PRIORITY_STREAM_REPLY structure is:
```
00h  A_UINT8  1   status;            ;\
01h  A_UINT8  1   txQueueNumber;     ; same as WMI_CRE_PRIORITY_STREAM_REPLY
02h  A_UINT8  1   rxQueueNumber;     ;/
03h  A_UINT8  1   trafficDirection;  ;\unlike WMI_CRE_PRIORITY_STREAM_REPLY
04h  A_UINT8  1   trafficClass;      ;/(entries are swapped)
```

WMIevent(?) - WMI_FRAME_RATES_REPLY ;-not implemented in DSi
Unknown crap. WMI_FRAME_RATES_REPLY is said to have same structure as WMI_SET_FRAMERATES_CMD parameter structure. But WMIevent(?) number is unknown, and DSi doesn't seem to send any such REPLY.

WMIevent(101Bh) - WMI_PEER_NODE_EVENT
Event Data:
```
00h  A_UINT8  1  eventCode;
01h  A_UINT8  6  peerMacAddr[ATH_MAC_LEN];
```
Below PEER values are probably meant to be the "eventCode" values(?):
```
PEER_NODE_JOIN_EVENT       = 00h
PEER_NODE_LEAVE_EVENT      = 01h
PEER_FIRST_NODE_JOIN_EVENT = 10h
PEER_LAST_NODE_LEAVE_EVENT = 11h
```

WMIevent(101Dh) - WMI_DTIMEXPIRY_EVENT
Event Data:
```
Unknown (if any)
```

WMIevent(101Eh) - WMI_WLAN_VERSION_EVENT
Event Data:
```
00h  A_UINT32 4  version;
```
Whatever event with whatever version?

WMIevent(1023h) - WMI_TX_COMPLETE_EVENT
Event Data:

```
  00h  A_UINT8  1  numMessages ;number of tx comp msgs following
  01h  A_UINT8  1  msgLen   ;length in bytes for each individual msg following
  02h  A_UINT8  1  msgType  ;version of tx complete msg data following
  03h  A_UINT8  1  reserved
 When msgType=01h=WMI_TXCOMPLETE_VERSION_1
  04h  ...      .. individual message(s) (see TX_COMPLETE_MSG_V1 structure)
 When msgType=Other
  04h  ...      .. reserved for other MSG types (none such defined yet)
```

TX_COMPLETE_MSG_V1 structure:

```
  00h  A_UINT8 1   status      /* one of TX_COMPLETE_STATUS_xxx values
  01h  A_UINT8 1   pktID       /* packet ID to identify parent packet
  02h  A_UINT8 1   rateIdx     /* rate index on successful transmission
  03h  A_UINT8 1   ackFailures /* number of ACK failures in tx attempt
       #if 0  ;optional "host delivery time" params currently ommitted...
  --     A_UINT32  queueDelay  /* usec delay measured Tx Start time
  --     A_UINT32  mediaDelay  /* usec delay measured ACK rx time
       #endif
```

TX_COMPLETE_STATUS_xxx values:

```
  TX_COMPLETE_STATUS_SUCCESS = 0
  TX_COMPLETE_STATUS_RETRIES = 1
  TX_COMPLETE_STATUS_NOLINK  = 2
  TX_COMPLETE_STATUS_TIMEOUT = 3
  TX_COMPLETE_STATUS_OTHER   = 4
```

Transmit complete event.

WMIevent(1025h) - WMI_ACL_DATA_EVENT

Event Data:

```
  Unknown (what?)
```

ACL is what? Is that somehow related to "ACLCOEX"?

WMIevent(1026h, or formerly N/A, or N/A) - WMI_REPORT_SLEEP_STATE_EVENT

Event Data:

```
  00h  A_UINT32 4  sleepState;
```

Values for "sleepState":

```
  WMI_REPORT_SLEEP_STATUS_IS_DEEP_SLEEP = 0
  WMI_REPORT_SLEEP_STATUS_IS_AWAKE      = 1
```

Names: WMI_REPORT_SLEEP_STATE_EVENT aka WMI_REPORT_SLEEP_STATUS

WMIevent(1027h, or formerly 1026h, or N/A) - WMI_WAPI_REKEY_EVENT

This event is added/removed randomly in different source code versions.

Event Data:

```
  00h  A_UINT8  1   type;
  01h  A_UINT8  6   macAddr[ATH_MAC_LEN];
```

Values (probably for above "type" field?):

```
  WAPI_REKEY_UCAST = 1
  WAPI_REKEY_MCAST = 2
```

The numbering for WMIevent(1026h..1029h) does vary in older source versions because WMI_REPORT_SLEEP_STATE_EVENT originally didn't exist, and WMI_WAPI_REKEY_EVENT originally did exist only if "WAPI_ENABLE". Later source code did always include WMI_WAPI_REKEY_EVENT, and even later code did re-remove it completely.

Names: WMI_WAPI_REKEY_EVENT aka WMI_WAPIREKEY_EVENT

WMIevent(1035h) - WMI_CCX_RM_STATUS_EVENT ;CCX Evants, uh, EvAntS?

Event Data:

```
  00h  A_INT32 4  rm_type   ;\one of these MIGHT be "WMI_CCX_RM_STATUS_TYPE" ?
  04h  A_INT32 4  status    ;/
```

WMI_CCX_RM_STATUS_TYPE values (probably for the "rm_type" field?):

```
  WMI_CCX_RM_STATUS_UNKNOWN     = 0
  WMI_CCX_RM_REPORT_SENT        = 1
  WMI_CCX_RM_REFUSE_REPORT_SENT = 2
```

Uh, "CCX" means... What?
Uh, "RM" means... maybe "radio_measurements" or What?
Maybe this is somehow related to "WMI_ENABLE_RM_CMD"?

WMIevent(1045h) - WMI_SET_HOST_SLEEP_MODE_CMD_PROCESSED_EVENT
Event Data:
```
  Unknown (if any?)
```
This event exists ONLY in source code from 2010, not in older code, and it's removed in newer code from 2012.
Special event used to notify host that AR6003 has processed sleep command (aka WMI_SET_HOST_SLEEP_MODE_CMD?) (needed to prevent a late incoming credit report from crashing the system).

WMIevent(9003h) - WMI_REPORT_WMM_PARAMS_EVENT
Event Data:
```
  00h  wmm_params  6*4  wmm_params[4];
```
"wmm_params" structure:
```
  00h  A_UINT8     1    acm;           /* ACM parameter
  01h  A_UINT8     1    aifsn;         /* AIFSN parameters
  02h  A_UINT8     1    logcwmin;      /* cwmin in exponential form
  03h  A_UINT8     1    logcwmax;      /* cwmax in exponential form
  04h  A_UINT16    2    txopLimit;     /* txopLimit
```

WMIcmd(?) - WMI_SET_ADHOC_BSSID_CMD
Parameters:
```
  00h  A_UINT8  6   bssid[ATH_MAC_LEN];
```
The above parameter structure is defined in "wmi.h", but there's no WMIcmd(xxxxh) command ID for it. Maybe the command did exist only in older versions (from dates before 2006)?

DSi Atheros Wifi - Unimplemented WMI Bluetooth Coexistence (newer AR6002)

/*----------COMMON to AR6002 and AR6003 ----------*/

BT_PARAMS_SCO_PSPOLL_LATENCY values:
```
  BT_PARAM_SCO_PSPOLL_LATENCY_ONE_FOURTH   = 1     ;aka 25%
  BT_PARAM_SCO_PSPOLL_LATENCY_HALF         = 2     ;aka 50%
  BT_PARAM_SCO_PSPOLL_LATENCY_THREE_FOURTH = 3     ;aka 75%
```

BT_PARAMS_SCO_STOMP_RULES values:
```
  BT_PARAMS_SCO_STOMP_SCO_NEVER      = 1
  BT_PARAMS_SCO_STOMP_SCO_ALWAYS     = 2
  BT_PARAMS_SCO_STOMP_SCO_IN_LOWRSSI = 3
```

BT_ANT_FRONTEND_CONFIG values:
```
  BT_ANT_TYPE_UNDEF         = 0   ;aka "Disabled (default)"
  BT_ANT_TYPE_DUAL          = 1
  BT_ANT_TYPE_SPLITTER      = 2
  BT_ANT_TYPE_SWITCH        = 3
  BT_ANT_TYPE_HIGH_ISO_DUAL = 4    ;<-- not in "code aurora"
```

BT_COLOCATED_DEV_TYPE values:
```
  BT_COLOCATED_DEV_BTS4020  = 0
  BT_COLCATED_DEV_CSR       = 1
  BT_COLOCATED_DEV_VALKYRIE = 2   ;aka BT_COLOCATED_DEV_VALKYRIe
```

/********** Applicable to AR6002 ONLY **********/

WMIcmd(003Bh) - WMI_SET_BT_STATUS_CMD ;AR6002 Bluetooth Coexistence only?

Parameters (02h bytes):

```
  00h  A_UINT8  1   streamType;         ;aka BT_STREAM_TYPE ?
  01h  A_UINT8  1   status;             ;aka BT_STREAM_STATUS ?
```

BT_STREAM_TYPE values:

```
  BT_STREAM_UNDEF = 0
  BT_STREAM_SCO   = 1 ;SCO stream
  BT_STREAM_A2DP  = 2 ;A2DP stream
  BT_STREAM_SCAN  = 3 ;BT Discovery or Page ;\"Newer AR6002 from 2008-2010"
  BT_STREAM_ESCO  = 4 ;Whatever             ;/
  BT_STREAM_ALL   = 5 ;Whatever             ;-"Newer AR6002 from 2008 only"
```

BT_STREAM_STATUS values:

```
  BT_STATUS_UNDEF        = 0
  BT_STATUS_START        = 1  ;-renamed to BT_STATUS_ON in code from 2010
  BT_STATUS_STOP         = 2  ;-renamed to BT_STATUS_OFF in code from 2010
  BT_STATUS_RESUME       = 3  ;\defined in "Older/Newer AR6002 from 2008"
  BT_STATUS_SUSPEND      = 4  ;/(not in "Newer AR6002 for 2010")
  BT_STATUS_SUSPEND_A2DP = 5  ;\defined in "Newer AR6002 from 2008")
  BT_STATUS_SUSPEND_SCO  = 6  ; (not in "Older AR6002 for 2008")
  BT_STATUS_SUSPEND_ACL  = 7  ; (not in "Newer AR6002 for 2010")
  BT_STATUS_SUSPEND_SCAN = 8  ;/
```

AR6002 only? (although other comment says "COMMON to AR6002 and AR6003"?)

WMIcmd(003Ch) - WMI_SET_BT_PARAMS_CMD ;AR6002 Bluetooth Coexistence only?

Parameters (1Fh or 19h bytes, for "Newer AR6002" code from 2008 or 2010):

```
  1Fh 19h  <--- total size (1Fh for code from 2008, 19h for code from 2010)
 when paramType=1=BT_PARAM_SCO:  ;SCO stream parameters (BT_PARAMS_SCO)
  00h 00h A_UINT32 4  numScoCyclesForceTrigger (Number SCO cycles after which
                       force a pspoll, default=10)
  04h 04h A_UINT32 4  dataResponseTimeout (Timeout Waiting for Downlink pkt in
                       response for ps-poll, default=10 ms)
  08h 08h A_UINT32 4  stompScoRules    ;aka BT_PARAMS_SCO_STOMP_RULES ?
  0Ch 0Ch A_UINT32 4  scoOptFlags (SCO Options Flags)
  10h --  A_UINT32 4  p2lrpOptModeBound     ;\PacketToLowRatePacketRatio's
  14h --  A_UINT32 4  p2lrpNonOptModeBound  ;/
  18h 10h A_UINT8  1  stompDutyCyleVal (SCO cycles to limit ps-poll queuing
                       if stomped)
  19h 11h A_UINT8  1  stompDutyCyleMaxVal (firmware increases stomp duty cycle
                       gradually uptill this value on need basis)
  1Ah 12h A_UINT8  1  psPollLatencyFraction (Fraction of idle period, within
                       which additional ps-polls can be queued)
  1Bh 13h A_UINT8  1  noSCOSlots (Number of SCO Tx/Rx slots. HVx,EV3,2EV3=2)
  1Ch 14h A_UINT8  1  noIdleSlots (Number of Bluetooth idle slots between
                       consecutive SCO Tx/Rx slots. HVx,EV3=4, 2EV3=10)
  1Dh --  A_UINT8  1  reserved8 (maintain word algnment) (uh, really?)
  --  15h A_UINT8  1  scoOptOffRssi (RSSI value below which we go to ps poll)
  --  16h A_UINT8  1  scoOptOnRssi (RSSI value above which we reenter opt mode)
  --  17h A_UINT8  1  scoOptRtsCount
 when paramType=2=BT_PARAM_A2DP: ;whatever (BT_PARAMS_A2DP)
  00h 00h A_UINT32 4  a2dpWlanUsageLimit (MAX time firmware uses the medium for
                       wlan, after it identifies the idle time, default=30 ms)
  04h 04h A_UINT32 4  a2dpBurstCntMin (Minimum number of bluetooth data frames
                       to replenish Wlan Usage limit, default 3)
  08h 08h A_UINT32 4  a2dpDataRespTimeout
  0Ch 0Ch A_UINT32 4  a2dpOptFlags (A2DP Option flags)
  10h --  A_UINT32 4  p2lrpOptModeBound     ;\PacketToLowRatePacketRatio's
  14h --  A_UINT32 4  p2lrpNonOptModeBound  ;/
  18h --  A_UINT16 2  reserved16 (maintain word alignment)
  1Ah 10h A_UINT8  1  isCoLocatedBtRoleMaster
  1Bh --  A_UINT8  1  reserved8 (maintain word alignment)
  1Ch --  PAD      2  undefined/padding
  --  11h A_UINT8  1  a2dpOptOffRssi (RSSI value below which we go to ps poll)
```

```
  --  12h A_UINT8  1  a2dpOptOnRssi(RSSI value above which we reenter opt mode)
  --  13h A_UINT8  1  a2dpOptRtsCount
  --  14h PAD      4  undefined/padding
 when paramType=3=BT_PARAM_ANTENNA_CONFIG:
  00h 00h A_UINT8  1   antType          aka BT_ANT_FRONTEND_CONFIG
  01h --  PAD      1Dh undefined/padding
  --  01h PAD      17h undefined/padding
 when paramType=4=BT_PARAM_COLOCATED_BT_DEVICE:
  00h 00h A_UINT8  1   coLocatedBtDev   aka BT_COLOCATED_DEV_TYPE
  01h --  PAD      1Dh undefined/padding
  --  01h PAD      17h undefined/padding
 when paramType=5=BT_PARAM_ACLCOEX: ;whatever (BT_PARAMS_ACLCOEX)
  ;During BT ftp/ BT OPP or any another data based acl profile on bluetooth
  ;(non a2dp).
  00h 00h A_UINT32 4   aclWlanMediumUsageTime (Wlan usage time during
                        Acl (non-a2dp) coexistence, default=30 ms)
  04h 04h A_UINT32 4   aclBtMediumUsageTime (Bt usage time during
                        acl coexistence, default=30 ms)
  08h 08h A_UINT32 4   aclDataRespTimeout
  0Ch 0Ch A_UINT32 4   aclDetectTimeout (ACL coexistence enabled if we get
                        10 Pkts in X ms, default=100 ms)
  10h 10h A_UINT32 4   aclmaxPktCnt (No of ACL pkts to receive before
                        enabling ACL coex)
  14h --  PAD      0Ah undefined/padding
  --  14h PAD      4   undefined/padding
 when paramType=6=BT_PARAM_11A_SEPARATE_ANT:
  00h 00h UNKNOWN  ?   unknown (maybe same as antType ?)
  xxh --  PAD      ..  undefined/padding
  --  xxh PAD      ..  undefined/padding
 and, in all cases:
  1Eh 18h A_UINT8  1   paramType
```

Values for "scoOptFlags" and "a2dpOptFlags":

```
  Bit0       Allow Close Range Optimization             ;\all versions
  Bit1       Force awake during close range             ;/
  Bit2       If set use (host supplied) threshold       ;\Newer AR6002
  Bit3..23   Unused                                     ;/from 2008
  Bit2       If set use host supplied RSSI for OPT      ;\
  Bit3       If set use host supplied RTS COUNT for OPT ; Newer AR6002
  Bit4..7    Unused                                     ; from 2010
  Bit8..15   Low Data Rate Min Cnt                      ;
  Bit16..23  Low Data Rate Max Cnt                      ;/
  Bit24..31  Undocumented (unused?)                     ;-all versions
```

PacketToLowRatePacketRatio's (p2lrp) entries (in code from 2008 only):

```
  p2lrpOptModeBound:    Minimum ratio required to STAY IN opt mode
  p2lrpNonOptModeBound: Minimum ratio required to SWITCH TO opt mode
```

DSi Atheros Wifi - Unimplemented WMI Bluetooth Coexistence (AR6003)

WMIcmd(003Bh) - WMI_SET_BT_STATUS_CMD ;AR6002 Bluetooth Coexistence only?
This command does maybe exist for AR6003 too (conflicting comments claim that it is for AR6002 only, or for both AR6002 and AR6003). See "Newer AR6002" description for details.
Also possible that it's replaced by "WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD".

WMIcmd(003Ch) - WMI_SET_BT_PARAMS_CMD ;AR6002 Bluetooth Coexistence only?
This command is used for AR6002 only. On AR6003, it's replaced by the new commands described below:

WMIcmd(F02Ah) - WMI_SET_BTCOEX_FE_ANT_CMD

Parameters:

```
  00h  A_UINT8  1  btcoexFeAntType
                    1 - WMI_BTCOEX_FE_ANT_SINGLE for single antenna front end
                    2 - WMI_BTCOEX_FE_ANT_DUAL for dual antenna front end
                        (for isolations less 35dB, for higher isolation there
                        is not need to pass this command).
                        (not implemented)
```

WMI_BTCOEX_FE_ANT_TYPE values:

```
  WMI_BTCOEX_NOT_ENABLED          = 0
  WMI_BTCOEX_FE_ANT_SINGLE        = 1
  WMI_BTCOEX_FE_ANT_DUAL          = 2
  WMI_BTCOEX_FE_ANT_DUAL_HIGH_ISO = 3
  WMI_BTCOEX_FE_ANT_BYPASS_MODE   = 4
  WMI_BTCOEX_FE_ANT_COMBINE_MODE  = 5
```

Indicates front end antenna configuration. This command needs to be issued right after initialization and after WMI_SET_BTCOEX_COLOCATED_BT_DEV_CMD. AR6003 enables coexistence and antenna switching based on the configuration.

WMIcmd(F02Bh) - WMI_SET_BTCOEX_COLOCATED_BT_DEV_CMD

Parameters:

```
  00h  A_UINT8  1  btcoexCoLocatedBTdev;   1 - Qcom BT (3 -wire PTA)
                                           2 - CSR BT  (3 wire PTA)
                                           3 - Atheros 3001 BT (3 wire PTA)
                                           4 - STE bluetooth (4-wire ePTA)
                                           5 - Atheros 3002 BT (4-wire MCI)
                                           default=3 (Atheros 3001 BT )
```

Indicate the bluetooth chip to the firmware. Firmware can have different algorithm based bluetooth chip type. Based on bluetooth device, different coexistence protocol would be used.

WMIcmd(F02Ch) - WMI_SET_BTCOEX_SCO_CONFIG_CMD

Parameters:

```
  --------------- BTCOEX_SCO_CONFIG scoConfig;
  00h  A_UINT32 4  scoSlots (Number of SCO Tx/Rx slots: HVx,EV3,2EV3 = 2)
  04h  A_UINT32 4  scoIdleSlots (Number of Bluetooth idle slots between
                    consecutive SCO Tx/Rx slots: HVx,EV3 = 4, 2EV3 = 10)
  08h  A_UINT32 4  scoFlags;  SCO Options Flags:
                              Bit0   Allow Close Range Optimization
                              Bit1   Is EDR capable or Not
                              Bit2   IS Co-located Bt role Master
                              Bit3   Firmware determines the periodicity of SCO
  0Ch  A_UINT32 4  linkId (applicable to STE-BT - not used)
  --------------- BTCOEX_PSPOLLMODE_SCO_CONFIG scoPspollConfig;
  10h  A_UINT32 4  scoCyclesForceTrigger (Number SCO cycles after which
                    force a pspoll, default=10)
  14h  A_UINT32 4  scoDataResponseTimeout (Timeout Waiting for Downlink pkt
                    in response for ps-poll, default=20 ms)
  18h  A_UINT32 4  scoStompDutyCyleVal (not implemented)
  1Ch  A_UINT32 4  scoStompDutyCyleMaxVal (not implemented)
  20h  A_UINT32 4  scoPsPollLatencyFraction (Fraction of idle period, within
                    which additional ps-polls can be queued
                      1 - 1/4 of idle duration
                      2 - 1/2 of idle duration
                      3 - 3/4 of idle duration
                      default=2 (1/2)
  --------------- BTCOEX_OPTMODE_SCO_CONFIG scoOptModeConfig;
  24h  A_UINT32 4  scoStompCntIn100ms (max number of SCO stomp in 100ms
                    allowed in opt mode. If exceeds the configured value,
                    switch to ps-poll mode, default=3)
  28h  A_UINT32 4  scoContStompMax (max number of continous stomp allowed in
                    opt mode. if excedded switch to pspoll mode, default=3)
  2Ch  A_UINT32 4  scoMinlowRateMbps (Low rate threshold)
```

```
  30h  A_UINT32 4  scoLowRateCnt (number of low rate pkts (< scoMinlowRateMbps)
                    allowed in 100 ms. If exceeded switch/stay to ps-poll mode,
                    lower stay in opt mode, default=36)
  34h  A_UINT32 4  scoHighPktRatio "(Total Rx pkts in 100 ms + 1)/((Total tx
                    pkts in 100 ms - No of high rate pkts in 100 ms) + 1) in
                    100 ms"
                    if exceeded switch/stay in opt mode and if lower
                    switch/stay in pspoll mode.
                    default=5 (80% of high rates)
  38h  A_UINT32 4  scoMaxAggrSize (Max number of Rx subframes allowed in this
                    mode. (Firmware re-negogiates max number of aggregates if
                    it was negogiated to higher value, default=1,
                    Recommended value Basic rate headsets = 1, EDR (2-EV3) =4.
  --------------- BTCOEX_WLANSCAN_SCO_CONFIG scoWlanScanConfig;
  3Ch  A_UINT32 4  scanInterval;
  40h  A_UINT32 4  maxScanStompCnt;
```

Configure SCO parameters. These parameters would be used whenever firmware is indicated of (e)SCO profile on bluetooth (via WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD).
Configration of BTCOEX_SCO_CONFIG data structure are common configuration and applies ps-poll mode and opt mode.
Ps-poll Mode - Station is in power-save and retrieves downlink data between sco gaps.
Opt Mode - station is in awake state and access point can send data to station any time.
BTCOEX_PSPOLLMODE_SCO_CONFIG - Configuration applied only during ps-poll mode.
BTCOEX_OPTMODE_SCO_CONFIG - Configuration applied only during opt mode.

```
  Aliases for "scoFlags":
  #define WMI_SCO_CONFIG_FLAG_ALLOW_OPTIMIZATION   (1 << 0)
  #define WMI_SCO_CONFIG_FLAG_IS_EDR_CAPABLE       (1 << 1)
  #define WMI_SCO_CONFIG_FLAG_IS_BT_MASTER         (1 << 2)
  #define WMI_SCO_CONFIG_FLAG_FW_DETECT_OF_PER     (1 << 3)
```

WMIcmd(F02Dh) - WMI_SET_BTCOEX_A2DP_CONFIG_CMD

Parameters:

```
  --------------- BTCOEX_A2DP_CONFIG a2dpConfig;
  00h  A_UINT32 4  a2dpFlags;  2DP Option flags:
                               Bit0   Allow Close Range Optimization
                               Bit1   IS EDR capable
                               Bit2   IS Co-located Bt role Master
                               Bit3   a2dp traffic is high priority
                               Bit4   Fw detect the role of bluetooth.
  04h  A_UINT32 4  linkId (Applicable only to STE-BT - not used)
  --------------- BTCOEX_PSPOLLMODE_A2DP_CONFIG a2dppspollConfig;
  08h  A_UINT32 4  a2dpWlanMaxDur (MAX time firmware uses the medium for
                    wlan, after it identifies the idle time, default=30 ms)
  0Ch  A_UINT32 4  a2dpMinBurstCnt (Minimum number of bluetooth data frames
                    to replenish Wlan Usage limit, default=3)
  10h  A_UINT32 4  a2dpDataRespTimeout (Max duration firmware waits for
                    downlink by stomping on  bluetooth after ps-poll is
                    acknowledged, default=20 ms)
  --------------- BTCOEX_OPTMODE_A2DP_CONFIG a2dpOptConfig;
  14h  A_UINT32 4  a2dpMinlowRateMbps (Low rate threshold)
  18h  A_UINT32 4  a2dpLowRateCnt (number of low rate pkts
                    (<a2dpMinlowRateMbps) allowed in 100 ms.
                    If exceeded switch/stay to ps-poll mode, lower stay in
                    opt mode, default=36)
  1Ch  A_UINT32 4  a2dpHighPktRatio "(Total Rx pkts in 100 ms + 1)/
                    ((Total tx pkts in 100 ms - No of high rate pkts in 100 ms)
                    + 1) in 100 ms",    if exceeded switch/stay in opt mode and
                    if lower switch/stay in pspoll mode.
                    default=5 (80% of high rates)
  20h  A_UINT32 4  a2dpMaxAggrSize (Max number of Rx subframes allowed in this
                    mode. (Firmware re-negogiates max number of aggregates if
```

```
                  it was negogiated to higher value, default=1.
                  Recommended value Basic rate headsets = 1, EDR (2-EV3) =8)
  24h  A_UINT32 4  a2dpPktStompCnt (number of a2dp pkts that can be stomped
                  per burst, default=6)
```

Configure A2DP profile parameters. These parameters would be used whenver firmware is indicated of A2DP profile on bluetooth (via WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD).
Configuration of BTCOEX_A2DP_CONFIG data structure are common configuration and applies to ps-poll mode and opt mode.
Ps-poll Mode - Station is in power-save and retrieves downlink data between a2dp data bursts.
Opt Mode - station is in power save during a2dp bursts and awake in the gaps.
BTCOEX_PSPOLLMODE_A2DP_CONFIG - Configuration applied only during ps-poll mode.
BTCOEX_OPTMODE_A2DP_CONFIG - Configuration applied only during opt mode.

```
  Aliases for "a2dpFlags":
  #define WMI_A2DP_CONFIG_FLAG_ALLOW_OPTIMIZATION    (1 << 0)
  #define WMI_A2DP_CONFIG_FLAG_IS_EDR_CAPABLE        (1 << 1)
  #define WMI_A2DP_CONFIG_FLAG_IS_BT_ROLE_MASTER     (1 << 2)
  #define WMI_A2DP_CONFIG_FLAG_IS_A2DP_HIGH_PRI      (1 << 3)
  #define WMI_A2DP_CONFIG_FLAG_FIND_BT_ROLE          (1 << 4)
```

WMIcmd(F02Eh) - WMI_SET_BTCOEX_ACLCOEX_CONFIG_CMD

Parameters:

```
  --------------- BTCOEX_ACLCOEX_CONFIG aclCoexConfig;
  00h A_UINT32 4  aclWlanMediumDur (Wlan usage time during Acl (non-a2dp)
                   coexistence, default=30 ms)
  04h A_UINT32 4  aclBtMediumDur (Bt usage time during acl coexistence,
                   default=30 ms)
  08h A_UINT32 4  aclDetectTimeout (BT activity observation time limit.
                   In this time duration, number of bt pkts are counted.
                   If the Cnt reaches "aclPktCntLowerLimit" value for
                   "aclIterToEnableCoex" iteration continuously, firmware gets
                   into ACL coexistence mode. Similarly, if bt traffic count
                   during ACL coexistence has not reached "aclPktCntLowerLimit"
                   continuously for "aclIterToEnableCoex", then ACL coexistence
                   is disabled, default=100 ms)
  0Ch A_UINT32 4  aclPktCntLowerLimit (Acl Pkt Cnt to be received in duration
                   of "aclDetectTimeout" for "aclIterForEnDis" times to
                   enabling ACL coex. Similar logic is used to disable acl
                   coexistence. (If "aclPktCntLowerLimit" cnt of acl pkts
                   are not seen by the for "aclIterForEnDis" then acl
                   coexistence is disabled), default=10)
  10h  A_UINT32 4  aclIterForEnDis (number of Iteration of
                   "aclPktCntLowerLimit" for Enabling and Disabling Acl
                   Coexistence, default=3)
  14h A_UINT32 4  aclPktCntUpperLimit (This is upperBound limit, if there is
                   more than "aclPktCntUpperLimit" seen in "aclDetectTimeout",
                   ACL coexistence is enabled right away, default=15)
  18h A_UINT32 4  aclCoexFlags   A2DP Option flags:
                                 Bit0  Allow Close Range Optimization
                                 Bit1  disable Firmware detection
                      (Currently supported configuration is aclCoexFlags=0)
  1Ch A_UINT32 4  linkId;           ;Applicable only for STE-BT - not used
  --------------- BTCOEX_PSPOLLMODE_ACLCOEX_CONFIG aclCoexPspollConfig;
  20h  A_UINT32 4  aclDataRespTimeout (Max duration firmware waits for downlink
                   by stomping on bluetooth after ps-poll is acknowledged,
                   default=20 ms)
  --------------- BTCOEX_OPTMODE_ACLCOEX_CONFIG aclCoexOptConfig;
  24h  A_UINT32 4  aclCoexMinlowRateMbps        ;\
  28h  A_UINT32 4  aclCoexLowRateCnt            ;
  2Ch  A_UINT32 4  aclCoexHighPktRatio          ; Not implemented yet
  30h  A_UINT32 4  aclCoexMaxAggrSize           ;
  34h  A_UINT32 4  aclPktStompCnt               ;/
```

Configure non-A2dp ACL profile parameters.The starts of ACL profile can either be indicated via WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD or enabled via firmware detection which is configured via "aclCoexFlags".
Configration of BTCOEX_ACLCOEX_CONFIG data structure are common configuration and applies ps-poll mode and opt mode.
Ps-poll Mode - Station is in power-save and retrieves downlink data during wlan medium.
Opt Mode - station is in power save during bluetooth medium time and awake during wlan duration.

```
            (Not implemented yet)    (uh, what?)
```

BTCOEX_PSPOLLMODE_ACLCOEX_CONFIG - Configuration applied only during ps-poll mode.
BTCOEX_OPTMODE_ACLCOEX_CONFIG - Configuration applied only during opt mode.

```
  Aliases for "aclCoexFlags":
  #define WMI_ACLCOEX_FLAGS_ALLOW_OPTIMIZATION   (1 << 0)
  #define WMI_ACLCOEX_FLAGS_DISABLE_FW_DETECTION (1 << 1)
```

WMIcmd(F02Fh) - WMI_SET_BTCOEX_BTINQUIRY_PAGE_CONFIG_CMD
Parameters:

```
  00h  A_UINT32 4  btInquiryDataFetchFrequency (The frequency of querying the
                    AP for data (via pspoll) is configured by this parameter,
                    default=10 ms)
  04h  A_UINT32 4  protectBmissDurPostBtInquiry (The firmware will continue to
                    be in inquiry state for configured duration, after inquiry
                    completion. This is to ensure other bluetooth transactions
                    (RDP, SDP profiles, link key exchange, etc.) goes through
                    smoothly without wifi stomping, default=10 secs)
  08h  A_UINT32 4  maxpageStomp (Applicable only for STE-BT interface.
                    Currently not used)
  0Ch  A_UINT32 4  btInquiryPageFlag (Not used)
```

Configuration parameters during bluetooth inquiry and page. Page configuration is applicable only on interfaces which can distinguish page (applicable only for ePTA - STE bluetooth).
Bluetooth inquiry start and end is indicated via WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD. During this the station will be power-save mode.

WMIcmd(F030h) - WMI_SET_BTCOEX_DEBUG_CMD
Parameters:

```
  00h  A_UINT32 4  btcoexDbgParam1      ;\
  04h  A_UINT32 4  btcoexDbgParam2      ; Used for firmware development
  08h  A_UINT32 4  btcoexDbgParam3      ; and debugging
  0Ch  A_UINT32 4  btcoexDbgParam4      ;
  10h  A_UINT32 4  btcoexDbgParam5      ;/
```

WMIcmd(F031h) - WMI_SET_BTCOEX_BT_OPERATING_STATUS_CMD
Parameters:

```
  00h  A_UINT32 4  btProfileType (1=SCO, 2=A2DP, 3=INQUIRY_PAGE, 4=ACLCOEX)
  04h  A_UINT32 4  btOperatingStatus    ;aka BT_STREAM_STATUS on AR6002 ?
  08h  A_UINT32 4  btLinkId
```

WMI_BTCOEX_BT_PROFILE values:

```
  WMI_BTCOEX_BT_PROFILE_SCO          = 1
  WMI_BTCOEX_BT_PROFILE_A2DP         = 2
  WMI_BTCOEX_BT_PROFILE_INQUIRY_PAGE = 3
  WMI_BTCOEX_BT_PROFILE_ACLCOEX      = 4
```

This command is probably equivalent to WMI_SET_BT_STATUS_CMD on AR6002.

WMIcmd(F032h) - WMI_GET_BTCOEX_STATS_CMD ;reply WMI_REPORT_BTCOEX_STATS_EVENT
Parameters:

```
  Unknown (none?)
```

WMIcmd(F033h) - WMI_GET_BTCOEX_CONFIG_CMD ;reply

WMI_REPORT_BTCOEX_CONFIG_EV.
Parameters:
```
  00h  A_UINT32 4  btProfileType (1=SCO, 2=A2DP, 3=INQUIRY_PAGE, 4=ACLCOEX)
  04h  A_UINT32 4  linkId (not used) (reserved/dummy?)
```
Command to firmware to get configuration parameters of the bt profile reported via WMI_BTCOEX_CONFIG_EVENTID.

WMIevent(1029h, or formerly 1028h, or 1027h) - WMI_REPORT_BTCOEX_CONFIG_EVENT
Event Data:
```
  00h A_UINT32 4  btProfileType (1=SCO, 2=A2DP, 3=INQUIRY_PAGE, 4=ACLCOEX)
  04h A_UINT32 4  linkId (not used)
      PREPACK union -- below are same as parameters from corresponding CMD's:
  08h ..  WMI_SET_BTCOEX_SCO_CONFIG_CMD            scoConfigCmd;
  08h ..  WMI_SET_BTCOEX_A2DP_CONFIG_CMD           a2dpConfigCmd;
  08h ..  WMI_SET_BTCOEX_ACLCOEX_CONFIG_CMD        aclcoexConfig;
  08h ..  WMI_SET_BTCOEX_BTINQUIRY_PAGE_CONFIG_CMD btinquiryPageConfigCmd;
```
Event from firmware to host, sent in response to WMI_GET_BTCOEX_CONFIG_CMD.

WMIevent(1028h, or formerly 1027h, or 1026h) - WMI_REPORT_BTCOEX_STATS_EVENT
Event Data:
```
  ---------------  BTCOEX_GENERAL_STATS coexStats;
  00h A_UINT32 4  highRatePktCnt;
  04h A_UINT32 4  firstBmissCnt;
  08h A_UINT32 4  psPollFailureCnt;
  0Ch A_UINT32 4  nullFrameFailureCnt;
  10h A_UINT32 4  optModeTransitionCnt;
  ---------------  BTCOEX_SCO_STATS scoStats;
  14h A_UINT32 4  scoStompCntAvg;
  18h A_UINT32 4  scoStompIn100ms;
  1Ch A_UINT32 4  scoMaxContStomp;
  20h A_UINT32 4  scoAvgNoRetries;
  24h A_UINT32 4  scoMaxNoRetriesIn100ms;
  ---------------  BTCOEX_A2DP_STATS a2dpStats;
  28h A_UINT32 4  a2dpBurstCnt;
  2Ch A_UINT32 4  a2dpMaxBurstCnt;
  30h A_UINT32 4  a2dpAvgIdletimeIn100ms;
  34h A_UINT32 4  a2dpAvgStompCnt;
  ---------------  BTCOEX_ACLCOEX_STATS aclCoexStats;
  38h A_UINT32 4  aclPktCntInBtTime;
  3Ch A_UINT32 4  aclStompCntInWlanTime;
  40h A_UINT32 4  aclPktCntIn100ms;
```
Used for firmware development and debugging.
Names: WMI_GET_BTCOEX_STATS aka WMI_REPORT_BTCOEX_STATS_EVENT aka WMI_REPORT_BTCOEX_BTCOEX_STATS_EVENT

DSi Atheros Wifi - Unimplemented WMI DataSet Functions

Not implemented in DSi.

The "DataSet" feature allows the firmware to read external data from host memory for whatever purpose (maybe intended for cases where the wifi board doesn't contain an EEPROM, or for cases where Xtensa RAM/ROM is too small to hold the whole firmware, or just to make it easier to modify data for testing/debugging).
For the "DataSet" stuff, the normal Command/Response flow is reversed: the firmware does send an EVENT to request data, and the host must respond by a REPLY_CMD (except for the CLOSE_EVENT, which requires no reply).

WMIevent(1010h:3001h) - WMIX_DSETOPENREQ_EVENT
```
  00h  A_UINT32 4   dset_id          ;-ID of requested DataSet (see "dsetid.h")
```

```
  04h  A_UINT32 4   targ_dset_handle ;\to be echo'ed in REPLY_CMD
  08h  A_UINT32 4   targ_reply_fn    ; (host doesn't need to deal with this)
  0Ch  A_UINT32 4   targ_reply_arg   ;/
```
DataSet Open Request Event. The host should open the DataSet and send a WMIX_DSETOPEN_REPLY_CMD.

WMIevent(1010h:3003h) - WMIX_DSETDATAREQ_EVENT
```
  00h  A_UINT32 4   access_cookie    ;-some kind of "filehandle" on host side
  04h  A_UINT32 4   offset           ;\source offset & length of requested data
  08h  A_UINT32 4   length           ;/
  0Ch  A_UINT32 4   targ_buf         ;\to be echo'ed in REPLY_CMD
  10h  A_UINT32 4   targ_reply_fn    ; (host doesn't need to deal with this)
  14h  A_UINT32 4   targ_reply_arg   ;/
```
DataSet Data Request Event. The host should send the requested data via WMIX_DSETDATA_REPLY_CMD.

WMIevent(1010h:3002h) - WMIX_DSETCLOSE_EVENT
```
  00h  A_UINT32 4   access_cookie       ;-some kind of "filehandle" on host side
```
DataSet Close Event. The host should close the DataSet (and doesn't need to send any REPLY_CMD).

WMIcmd(002Eh:2001h) - WMIX_DSETOPEN_REPLY_CMD
```
  00h  A_UINT32 4   status            ;-what status ?
  04h  A_UINT32 4   targ_dset_handle  ;\
  08h  A_UINT32 4   targ_reply_fn     ; to be echo'ed from open EVENT
  0Ch  A_UINT32 4   targ_reply_arg    ;/
  10h  A_UINT32 4   access_cookie     ;-some kind of "filehandle" on host side
  14h  A_UINT32 4   size              ;-what size ?
  18h  A_UINT32 4   version           ;-what version ?
```
This REPLY_CMD should be send in response to WMIX_DSETOPENREQ_EVENTs.

WMIcmd(002Eh:2002h) - WMIX_DSETDATA_REPLY_CMD
```
  00h  A_UINT32 4   status            ;-what status ?
  04h  A_UINT32 4   targ_buf          ;\
  08h  A_UINT32 4   targ_reply_fn     ; to be echo'ed from data EVENT
  0Ch  A_UINT32 4   targ_reply_arg    ;/
  10h  A_UINT32 4   length            ;\requested data
  14h  A_UINT8  LEN buf[length]       ;/
```
This REPLY_CMD should be send in response to WMIX_DSETDATAREQ_EVENTs.

DSi Atheros Wifi - Unimplemented WMI AP Mode Functions (exists on 3DS)

AP Mode
Whatever that is... probably something where the AR600x acts as Access Point (AP) for other Stations (STA). The DSi doesn't support AP Mode stuff, however, 3DS firmware type4 does support most of them (but with F00Ah..F013h renumbered to 004Ah..0052h).

AP Mode definitions
Changing the following values needs compilation of both driver and firmware.
```
  AP_MAX_NUM_STA = 4   ;for old AR6002_REV2 version
  AP_MAX_NUM_STA = 10  ;for newer versions
  NUM_DEV        = 3   ;Maximum no. of virtual interface supported
  NUM_CONN       = (AP_MAX_NUM_STA + NUM_DEV)
  AP_ACL_SIZE         = 10
  IEEE80211_MAX_IE    = 256
  MCAST_AID           = 0FFh   ;Spl. AID used to set DTIM flag in the beacons
  DEF_AP_COUNTRY_CODE = "US "
  DEF_AP_WMODE_G      = WMI_11G_MODE
```

```
  DEF_AP_WMODE_AG     = WMI_11AG_MODE
  DEF_AP_DTIM         = 5
  DEF_BEACON_INTERVAL = 100
  AP_DISCONNECT_STA_LEFT     = 101  ;\
  AP_DISCONNECT_FROM_HOST    = 102  ;
  AP_DISCONNECT_COMM_TIMEOUT = 103  ; AP mode disconnect reasons
  AP_DISCONNECT_MAX_STA      = 104  ; (101..107 decimal):
  AP_DISCONNECT_ACL          = 105  ;
  AP_DISCONNECT_STA_ROAM     = 106  ;
  AP_DISCONNECT_DFS_CHANNEL  = 107  ;/
```

WMIcmd(F00Bh or 004Ah) - WMI_AP_HIDDEN_SSID_CMD

Parameters:

```
  00h  A_UINT8  1   hidden_ssid;
```

#define HIDDEN_SSID_FALSE = 0
#define HIDDEN_SSID_TRUE = 1

WMIcmd(F00Ch or 004Bh) - WMI_AP_SET_NUM_STA_CMD aka WMI_AP_NUM_STA_CMD

Parameters:

```
  00h  A_UINT8  1   num_sta;
```

WMIcmd(F00Dh or 004Ch) - WMI_AP_ACL_POLICY_CMD

Parameters:

```
  00h  A_UINT8  1   policy;
 #define AP_ACL_DISABLE          = 00h
 #define AP_ACL_ALLOW_MAC        = 01h
 #define AP_ACL_DENY_MAC         = 02h
 #define AP_ACL_RETAIN_LIST_MASK = 80h
```

WMIcmd(F00Eh or 004Dh) - WMI_AP_ACL_MAC_LIST_CMD aka WMI_AP_ACL_MAC_CMD

Parameters:

```
  00h  A_UINT8  1   action;
  01h  A_UINT8  1   index;
  02h  A_UINT8  6   mac[ATH_MAC_LEN];
  08h  A_UINT8  1   wildcard;
```

ADD_MAC_ADDR = 1
DEL_MAC_ADDR = 2

There is also a "WMI_AP_ACL" structure with unknown purpose (maybe internally used in Xtensa memory or whatever):

```
  A_UINT16 2   index;
  A_UINT8  ... acl_mac[AP_ACL_SIZE][ATH_MAC_LEN];
  A_UINT8  ..  wildcard[AP_ACL_SIZE];
  A_UINT8  1   policy;
```

WMIcmd(F00Fh or 004Eh) - WMI_AP_CONFIG_COMMIT_CMD

Unknown/undocumented? Except, see this:
sp.chip1stop.com/sp/wp-content/uploads/2015/05/gt202wifimoduleapiguidev13.pdf
Parameters: 34h bytes on 3DS.
Command Parameters: WMI_AP_CONFIG_COMMIT_CMD

```
  Type Name Comment
  00h  A_UINT8 networktype NETWORK_TYPE
         0x01 INFRA_NETWORK
         0x02 ADHOC_NETWORK
         0x04 ADHOC_CREATOR
  01h  A_UINT8 dot11authmode DOT11_AUTH_MODE
         0x01 OPEN_AUTH
         0x02 SHARED_AUTH
  02h  A_UINT8 authmode AUTH_MODE
         0x01 NONE_AUTH
         0x02 WPA_AUTH
```

```
        0x04 WPA2_AUTH
        0x08 WPA_PSK_AUTH
        0x10 WPA2_PSK_AUTH
        0x20 WPA_AUTH_CCKM
        0x40 WPA2_AUTH_CCKM
  03h  A_UINT8 pairwiseCryptoType CRYPTO_TYPE
        0x01 NONE_CRYPT
        0x02 WEP_CRYPT
        0x04 TKIP_CRYPT
        0x08 AES_CRYPT
  04h  A_UINT8 pairwiseCryptoLen Length in bytes. Valid when the type is
        WEP_CRYPT, otherwise this should be 0
  05h  A_UINT8 groupCryptoType CRYPTO_TYPE
  06h  A_UINT8 groupCryptoLen Length in bytes. Valid when the type is
        WEP_CRYPT, otherwise this should be 0
  07h  A_UINT8 ssidLength SSID length for the AP mode
  08h  A_UCHAR ssid[32] SSID value for the SoftAP mode
  28h  A_UINT16 channel Channel in which the AP mode has to be started
  2Ah  A_UINT8 bssid[6]
  30h  A_UINT8 ctrl_flags WMI_CONNECT_CTRL_FLAGS_BITS
    0x0001 CONNECT_ASSOC_POLICY_USER  Associative frames are sent using
           the policy specified by the CONNECT_SEND-_REASSOC flag
    0x0004 CONNECT_IGNORE_WPAx-_GROUP_CIPHER  Ignore the WPAx group cipher
           for WPA/WPA2
    0x0040 CONNECT_DO_WPA_OFFLOAD  Use the authenticator in the QCA4002(=chip)
    0x0100 CONNECT_WPS_FLAG   Set to indicate that the AP will add WPS IE to
           its beacon
    0xFFFF                    Reset all control flags
```

Uh, the flags are said to be 8bit... but can be as large as FFFFh?
Also, 3DS seems to use 34h bytes total... so flags would be 32bit maybe?
Whereof, 3DS seems to be simply ignoring param [2Ah..33h].

WMIcmd(F010h or 004Fh) - WMI_AP_SET_MLME_CMD

Parameters:

```
  00h  A_UINT8  6  mac[ATH_MAC_LEN];
  06h  A_UINT16 2  reason;              /* 802.11 reason code
  08h  A_UINT8  1  cmd;                 /* operation to perform
```

MLME Commands (aka above "cmd"):

```
  WMI_AP_MLME_ASSOC       1   /* associate station
  WMI_AP_DISASSOC         2   /* disassociate station
  WMI_AP_DEAUTH           3   /* deauthenticate station
  WMI_AP_MLME_AUTHORIZE   4   /* authorize station
  WMI_AP_MLME_UNAUTHORIZE 5   /* unauthorize station
```

WMIcmd(F011h or 0050h) - WMI_AP_SET_PVB_CMD

Parameters (06h bytes on 3DS, but other variant exists, too):

```
  06h 08h  <---- total size (on 3DS it's 06h)
  00h 00h  A_BOOL   4  flag;
  --  04h  A_UINT16 2  rsvd;
  04h 06h  A_UINT16 2  aid;
```

Note: The 3DS command list does support the 6-byte variant, but it seem to be simply ignoring the command & parameters.

WMIcmd(F012h or 0051h) - WMI_AP_CONN_INACT_CMD

Parameters:

```
  00h  A_UINT32 4  period;
```

WMIcmd(F013h or 0052h) - WMI_AP_PROT_SCAN_TIME_CMD

Parameters:

```
  00h  A_UINT32 4  period_min;
  04h  A_UINT32 4  dwell_ms;
```

_____ Below doesn't seem to exist on 3DS _____

WMIcmd(F014h) - WMI_AP_SET_COUNTRY_CMD ;aka formerly WMI_SET_COUNTRY_CMD
Parameters:
```
  00h  A_UCHAR  3  countryCode[3]      ;two letter ASCII... plus ending 00h?
```
WMI_DISABLE_REGULATORY_CODE = "FF" ;uh, are that THREE chars?

WMIcmd(F015h) - WMI_AP_SET_DTIM_CMD
Parameters:
```
  00h  A_UINT8  1  dtim;   ;wake up each N beacon interval units
```
DTIM interval used to check for multicast packets.

WMIcmd(F016h or N/A) - WMI_AP_MODE_STAT_CMD
Parameters:
```
  Unknown (if any)
```
Reply:
```
  Unknown (if any)
```
There is an "WMI_AP_MODE_STAT" structure (said to be an event, although there is no WMIevent(xxxxh) number defined for it anywhere) (and, the "action" entry sounds more like an command/parameter than like an event/response?):
WMI_AP_MODE_STAT structure:
```
  00h  A_UINT32            action;
  04h  WMI_PER_STA_STAT    sta[AP_MAX_NUM_STA];
```
Mysterious values (probably for the "action" entry:
```
  AP_GET_STATS   = 0
  AP_CLEAR_STATS = 1
```
WMI_PER_STA_STAT structure (for the "sta" entries):
```
  00h  A_UINT32 tx_bytes;
  04h  A_UINT32 tx_pkts;
  08h  A_UINT32 tx_error;
  0Ch  A_UINT32 tx_discard;
  10h  A_UINT32 rx_bytes;
  14h  A_UINT32 rx_pkts;
  18h  A_UINT32 rx_error;
  1Ch  A_UINT32 rx_discard;
  20h  A_UINT32 aid;
```

WMIcmd(F027h or N/A) - WMI_AP_SET_11BG_RATESET_CMD
Parameters:
```
  00h  A_UINT8  1  rateset;
 AP_11BG_RATESET1    = 1
 AP_11BG_RATESET2    = 2
 DEF_AP_11BG_RATESET = AP_11BG_RATESET1
```

WMIcmd(F05Ah or F059h or N/A) - WMI_AP_SET_APSD_CMD
Parameters:
```
  00h  A_UINT8  1  enable;
```
WMI_AP_APSD_DISABLED = 0
WMI_AP_APSD_ENABLED = 1

WMIcmd(F05Bh or F05Ah or N/A) - WMI_AP_APSD_BUFFERED_TRAFFIC_CMD
Parameters:
```
  00h  A_UINT16 2  aid;
  02h  A_UINT16 2  bitmap;
  04h  A_UINT32 4  flags;
```
WMI_AP_APSD_BUFFERED_TRAFFIC_FLAGS values (only one defined)
```
  WMI_AP_APSD_NO_DELIVERY_FRAMES_FOR_THIS_TRIGGER = 01h
```

WMIcmd(F06Eh or N/A) - WMI_AP_JOIN_BSS_CMD
Unknown/undocumented (some new command, invented in 2012 or so).

WMIevent(?) - WMI_AP_MODE_STAT_EVENT ;reply to WMI_AP_MODE_STAT_CMD ?
Maybe there is an event/reply for the "WMI_AP_MODE_STAT_CMD" command (see above).

WMIevent(101Ch) - WMI_PSPOLL_EVENT ;aka WMI_PS_POLL_EVENT ;AP mode related?
Event Data:
```
  00h  A_UINT16 2  aid;
```
Whatever. Said to be an "AP mode event". Maybe WMI_SET_AP_PS_CMD related?

WMIcmd(F0AFh) - WMI_AP_PSBUFF_OFFLOAD ;QCA4002 chipset on GT202 WiFi module
```
  00h  A_UINT8 Enable
         0 Enable buffering mechanism in firmware to handle power save clients
         1 Disable buffering mechanism
  01h  A_UINT8 psBufCount  Range: 1-3,
         Specifies number of buffers allowed to buffer power save packets.
```

DSi Atheros Wifi - Unimplemented WMI DFS Functions

WMIcmd(F034h) - WMI_SET_DFS_ENABLE_CMD ;aka WMI_SET_DFS_CMD maybe ?
WMIcmd(F035h) - WMI_SET_DFS_MINRSSITHRESH_CMD ;aka WMI_SET_DFS_CMD too ??
WMIcmd(F036h) - WMI_SET_DFS_MAXPULSEDUR_CMD ;aka WMI_SET_DFS_CMD too ??
Parameters:
```
  Unknown (maybe WMI_SET_DFS_CMD structure?)
```
There is a "WMI_SET_DFS_CMD" structure defined:
```
  00h  A_UINT8  1   enable;
```
Maybe that structure is meant to be used with one of the WMI_SET_DFS_xxx_CMD commands, or maybe it's even meant to be used with ALL of that THREE commands.
```
  XXX see file "dfs_common.h"
```

WMIcmd(F037h) - WMI_DFS_RADAR_DETECTED_CMD ;aka WMI_RADAR_DETECTED_CMD
Parameters:
```
  00h  A_UINT16 2  chan_index;
  02h  A_INT8   1  bang_radar;
```

WMIevent(102Bh) - WMI_DFS_HOST_ATTACH_EVENT
Event Data:
```
  00h  A_UINT64 8  ext_chan_busy_ts;
  08h  A_UINT8  1  enable_ar;
  09h  A_UINT8  1  enable_radar;
```

WMIevent(102Ch) - WMI_DFS_HOST_INIT_EVENT
Event Data:
```
  00h  A_UINT32 4  dfs_domain;
```

WMIevent(102Dh) - WMI_DFS_RESET_DELAYLINES_EVENT
WMIevent(102Eh) - WMI_DFS_RESET_RADARQ_EVENT
WMIevent(102Fh) - WMI_DFS_RESET_AR_EVENT
WMIevent(1030h) - WMI_DFS_RESET_ARQ_EVENT
WMIevent(1031h) - WMI_DFS_SET_DUR_MULTIPLIER_EVENT
WMIevent(1032h) - WMI_DFS_SET_BANGRADAR_EVENT
WMIevent(1033h) - WMI_DFS_SET_DEBUGLEVEL_EVENT
Event Data:

```
  Unknown (if any) (not defined in file "dfs_common.h")
```

WMIevent(1034h) - WMI_DFS_PHYERR_EVENT

Event Data:

```
  00h  A_UINT8         1     num_events;
  01h  dfs_event_info  ..    ev_info[WMI_DFS_EVENT_MAX_BUFFER_SIZE];
```

WMI_DFS_EVENT_MAX_BUFFER_SIZE = ((255)/sizeof(struct dfs_event_info))

Format of the above "dfs_event_info" structure:

```
  00h  A_UINT64 8  full_ts;       /* 64-bit full timestamp from interrupt time
  08h  A_UINT32 4  ts;            /* Original 15 bit recv timestamp
  0Ch  A_UINT32 4  ext_chan_busy; /* Ext chan busy %
  10h  A_UINT8  1  rssi;          /* rssi of radar event
  11h  A_UINT8  1  dur;           /* duration of radar pulse
  12h  A_UINT8  1  chanindex;     /* Channel of event
  13h  A_UINT8  1  flags;
```

Values for "flags":

```
  PRIMARY_CH = 0  ;\flags.bit0
  EXT_CH     = 1  ;/
  AR_EVENT   = 0  ;\flags.bit1
  DFS_EVENT  = 2  ;/
```

Some more DFS related constants (unknown purpose):

```
  DFS_UNINIT_DOMAIN = 0     ;Uninitialized dfs domain
  DFS_FCC_DOMAIN    = 1     ;FCC3 dfs domain
  DFS_ETSI_DOMAIN   = 2     ;ETSI dfs domain
  DFS_MKK4_DOMAIN   = 3     ;Japan dfs domain
  MAX_BIN5_DUR      = 131   ;rounded from 131.25=(105*1.25)   ;DFS related
  TRAFFIC_DETECTED = 1      ;whatever                        ;DFS related
  ATH_DEBUG_DFS  = 00000100h  ;Minimal DFS debug        ;\
  ATH_DEBUG_DFS1 = 00000200h  ;Normal DFS debug         ; should match the
  ATH_DEBUG_DFS2 = 00000400h  ;Maximal DFS debug        ; table from if_ath.c
  ATH_DEBUG_DFS3 = 00000800h  ;matched filterID display ;/
```

DSi Atheros Wifi - Unimplemented WMI P2P Functions

P2P module definitions

P2P_SSID structure:

```
  00h  A_UINT8  1    ssidLength;
  01h  A_UINT8  20h  ssid[WMI_MAX_SSID_LEN];
```

WMIcmd(F038h) - WMI_P2P_SET_CONFIG_CMD

Parameters:

```
  00h  A_UINT8  1  go_intent;
  01h  A_UINT8  3  country[3];
  04h  A_UINT8  1  reg_class;
  05h  A_UINT8  1  listen_channel;
  06h  A_UINT8  1  op_reg_class;
  07h  A_UINT8  1  op_channel;
  09h  A_UINT16 2  config_methods;
```

WMIcmd(F039h) - WMI_WPS_SET_CONFIG_CMD ;P2P related

Parameters:

```
  00h  device_type_tuple 4     pri_dev_type;
  --   outcommented?     0     //A_UINT8 pri_device_type[8];
  04h  device_type_tuple 4*5   sec_dev_type[MAX_P2P_SEC_DEVICE_TYPES];
  18h  A_UINT8           10h   uuid[WPS_UUID_LEN];
  28h  A_UINT8           20h   device_name[WPS_MAX_DEVNAME_LEN];
  48h  A_UINT8           1     dev_name_len;
```

"device_type_tuple" structure (4 bytes):

```
  00h  A_UINT16 2  categ;
  02h  A_UINT16 2  sub_categ;
```
MAX_P2P_SEC_DEVICE_TYPES = 5
WPS_UUID_LEN = 16
WPS_MAX_DEVNAME_LEN = 32

WMIcmd(F03Ah) - WMI_SET_REQ_DEV_ATTR_CMD ;P2P related
Parameters:
```
  00h  device_type_tuple 4     pri_dev_type;
  04h  device_type_tuple 4*5   sec_dev_type[MAX_P2P_SEC_DEVICE_TYPES];
  18h  A_UINT8           6     device_addr[ATH_MAC_LEN];
```

WMIcmd(F03Bh) - WMI_P2P_FIND_CMD
Parameters:
typedef PREPACK struct {
```
  00h  A_UINT32 4     timeout;
  04h  A_ENUM   ..    type;  ;A_UINTx or so?  ;aka WMI_P2P_DISC_TYPE
```
WMI_P2P_DISC_TYPE values:
```
  WMI_P2P_FIND_START_WITH_FULL = Unknown (0 or 1 or so)
  WMI_P2P_FIND_ONLY_SOCIAL     = WMI_P2P_FIND_START_WITH_FULL+1
  WMI_P2P_FIND_PROGRESSIVE     = WMI_P2P_FIND_START_WITH_FULL+2
```

WMIcmd(F03Ch) - WMI_P2P_STOP_FIND_CMD
Parameters:
```
  Unknown (none?)
```

WMIcmd(F03Dh) - WMI_P2P_GO_NEG_START_CMD
Parameters:
```
  00h  A_UINT16 2   listen_freq;
  02h  A_UINT16 2   force_freq;
  04h  A_UINT16 2   go_oper_freq;
  06h  A_UINT8  1   dialog_token;
  07h  A_UINT8  6   peer_addr[ATH_MAC_LEN];
  0Dh  A_UINT8  6   own_interface_addr[ATH_MAC_LEN];
  13h  A_UINT8  6   member_in_go_dev[ATH_MAC_LEN];
  19h  A_UINT8  1   go_dev_dialog_token;
  1Ah  P2P_SSID 21h peer_go_ssid;
  3Bh  A_UINT8  1   wps_method;
  3Ch  A_UINT8  1   dev_capab;
  3Dh  A_INT8   1   go_intent;
  3Eh  A_UINT8  1   persistent_grp;
```

WMIcmd(F03Eh) - WMI_P2P_LISTEN_CMD
Parameters:
```
  00h  A_UINT32 4   timeout;
```

WMIcmd(F050h or F053h) - WMI_P2P_GO_NEG_REQ_RSP_CMD
Parameters:
```
  000h  A_UINT16 2    listen_freq;
  002h  A_UINT16 2    force_freq;
  004h  A_UINT8  1    status;
  005h  A_INT8   1    go_intent;
  006h  A_UINT8  200h wps_buf[512];
  206h  A_UINT16 2    wps_buflen;
  208h  A_UINT8  200h p2p_buf[512];
  408h  A_UINT16 2    p2p_buflen;
  40Ah  A_UINT8  1    dialog_token;
  40Bh  A_UINT8  1    wps_method;
  40Ch  A_UINT8  1    persistent_grp;
  40Dh  A_UINT8  6    sa[ATH_MAC_LEN];
```

WMIcmd(F051h or F054h) - WMI_P2P_GRP_INIT_CMD

Parameters:

```
  00h  A_UINT8  1  persistent_group;
  01h  A_UINT8  1  group_formation;
```

WMIcmd(F052h or F055h) - WMI_P2P_GRP_FORMATION_DONE_CMD

Parameters:

```
  00h  A_UINT8  6  peer_addr[ATH_MAC_LEN];
  06h  A_UINT8  1  grp_formation_status;
```

WMIcmd(F053h or F056h) - WMI_P2P_INVITE_CMD

Parameters:

```
  00h  A_ENUM   ..   role;   ;A_UINTx or so?  ;WMI_P2P_INVITE_ROLE
  ..   A_UINT16 2    listen_freq;
  ..   A_UINT16 2    force_freq;
  ..   A_UINT8  1    dialog_token;
  ..   A_UINT8  6    peer_addr[ATH_MAC_LEN];
  ..   A_UINT8  6    bssid[ATH_MAC_LEN];
  ..   A_UINT8  6    go_dev_addr[ATH_MAC_LEN];
  ..   P2P_SSID 21h  ssid;
  ..   A_UINT8  1    is_persistent;
  ..   A_UINT8  1    wps_method;
```

WMI_P2P_INVITE_ROLE values:

```
  WMI_P2P_INVITE_ROLE_GO        = Unknown (0 or 1 or so)
  WMI_P2P_INVITE_ROLE_ACTIVE_GO = WMI_P2P_INVITE_ROLE_GO+1
  WMI_P2P_INVITE_ROLE_CLIENT    = WMI_P2P_INVITE_ROLE_GO+2
```

WMIcmd(F054h or F057h) - WMI_P2P_INVITE_REQ_RSP_CMD

Parameters:

```
  000h  A_UINT16 2    force_freq;
  002h  A_UINT8  1    status;
  003h  A_UINT8  1    dialog_token;
  004h  A_UINT8  200h p2p_buf[512];
  204h  A_UINT16 2    p2p_buflen;
  206h  A_UINT8  1    is_go;
  207h  A_UINT8  6    group_bssid[ATH_MAC_LEN];
```

WMIcmd(F055h or F058h) - WMI_P2P_PROV_DISC_REQ_CMD

Parameters:

```
  00h  A_UINT16 2    wps_method;
  02h  A_UINT16 2    listen_freq;
  04h  A_UINT8  1    dialog_token;
  05h  A_UINT8  6    peer[ATH_MAC_LEN];
  0Bh  A_UINT8  6    go_dev_addr[ATH_MAC_LEN];
  11h  P2P_SSID 21h  go_oper_ssid;
```

WMIcmd(F056h or F059h) - WMI_P2P_SET_CMD

Parameters:

```
  00h  A_UINT8  1    config_id;           ;set to one of WMI_P2P_CONF_ID
 When config_id=1=WMI_P2P_CONFID_LISTEN_CHANNEL  ;WMI_P2P_LISTEN_CHANNEL
  01h  A_UINT8  1    reg_class;
  02h  A_UINT8  1    listen_channel;
 When config_id=2=WMI_P2P_CONFID_CROSS_CONNECT   ;WMI_P2P_SET_CROSS_CONNECT
  01h  A_UINT8  1    flag;
 When config_id=3=WMI_P2P_CONFID_SSID_POSTFIX    ;WMI_P2P_SET_SSID_POSTFIX
  01h  A_UINT8  17h  ssid_postfix[WMI_MAX_SSID_LEN-9];
  18h  A_UINT8  1    ssid_postfix_len;
 When config_id=4=WMI_P2P_CONFID_INTRA_BSS       ;WMI_P2P_SET_INTRA_BSS
  01h  A_UINT8  1    flag;
 When config_id=5=WMI_P2P_CONFID_CONCURRENT_MODE ;WMI_P2P_SET_CONCURRENT_MODE
```

```
  01h  A_UINT8  1    flag;
 When config_id=6=WMI_P2P_CONFID_GO_INTENT       ;WMI_P2P_SET_GO_INTENT
  01h  A_UINT8  1    value;
 When config_id=7=WMI_P2P_CONFID_DEV_NAME        ;WMI_P2P_SET_DEV_NAME
  01h  A_UINT8  20h  dev_name[WPS_MAX_DEVNAME_LEN];
  21h  A_UINT8  1    dev_name_len;
```

WMIcmd(F05Bh or F05Ch) - WMI_P2P_SDPD_TX_CMD

Parameters:

```
  000h  A_UINT8  1    type;
  001h  A_UINT8  1    dialog_token;
  002h  A_UINT8  1    frag_id;
  003h  A_UINT8  1    reserved1;          /* alignment
  004h  A_UINT8  6    peer_addr[ATH_MAC_LEN];
  00Ah  A_UINT16 2    freq;
  00Ch  A_UINT16 2    status_code;
  00Eh  A_UINT16 2    comeback_delay;
  010h  A_UINT16 2    tlv_length;
  012h  A_UINT16 2    update_indic;
  014h  A_UINT16 2    total_length;
  016h  A_UINT16 2    reserved2;         /* future
  018h  A_UINT8  400h tlv[WMI_P2P_MAX_TLV_LEN];
```

WMI_P2P_SDPD_TYPE values:

```
  WMI_P2P_SD_TYPE_GAS_INITIAL_REQ   = 01h
  WMI_P2P_SD_TYPE_GAS_INITIAL_RESP  = 02h
  WMI_P2P_SD_TYPE_GAS_COMEBACK_REQ  = 03h
  WMI_P2P_SD_TYPE_GAS_COMEBACK_RESP = 04h
  WMI_P2P_PD_TYPE_RESP              = 05h
  WMI_P2P_SD_TYPE_STATUS_IND        = 06h
```

WMI_P2P_SDPD_TRANSACTION_STATUS values:

```
  WMI_P2P_SDPD_TRANSACTION_PENDING  = 01h
  WMI_P2P_SDPD_TRANSACTION_COMP     = 02h
```

WMI_P2P_MAX_TLV_LEN = 1024

WMIcmd(F05Ch or F05Dh) - WMI_P2P_STOP_SDPD_CMD

Parameters:

```
  Unknown (none?)
```

WMIcmd(F05Dh or F05Eh) - WMI_P2P_CANCEL_CMD

Parameters:

```
  Unknown (none?)
```

WMIcmd(F06Dh or N/A) - WMI_GET_P2P_INFO_CMD

Unknown/undocumented (invented around 2012).

...P2P Events...

WMIevent(1036h) - WMI_P2P_GO_NEG_RESULT_EVENT

Event Data:

```
  00h  A_UINT16 2   freq;
  02h  A_INT8   1   status;
  03h  A_UINT8  1   role_go;
  04h  A_UINT8  20h ssid[WMI_MAX_SSID_LEN];
  24h  A_UINT8  1   ssid_len;
  25h  A_CHAR   9   pass_phrase[WMI_MAX_PASSPHRASE_LEN];
  2Eh  A_UINT8  6   peer_device_addr[ATH_MAC_LEN];
  34h  A_UINT8  6   peer_interface_addr[ATH_MAC_LEN];
  3Ah  A_UINT8  1   wps_method;
  3Bh  A_UINT8  1   persistent_grp;
```

WMI_MAX_PASSPHRASE_LEN = 9

WMIevent(103Dh) - WMI_P2P_GO_NEG_REQ_EVENT

Event Data:

```
000h  A_UINT8  6    sa[ATH_MAC_LEN];
006h  A_UINT8  200h wps_buf[512];
206h  A_UINT16 2    wps_buflen;
208h  A_UINT8  200h p2p_buf[512];
408h  A_UINT16 2    p2p_buflen;
40Ah  A_UINT8  1    dialog_token;
```

WMIevent(103Eh) - WMI_P2P_INVITE_REQ_EVENT

Event Data:

```
000h  A_UINT8  200h p2p_buf[512];
200h  A_UINT16 2    p2p_buflen;
202h  A_UINT8  6    sa[ATH_MAC_LEN];
208h  A_UINT8  6    bssid[ATH_MAC_LEN];
20Eh  A_UINT8  6    go_dev_addr[ATH_MAC_LEN];
214h  P2P_SSID 21h  ssid;
235h  A_UINT8  1    is_persistent;
236h  A_UINT8  1    dialog_token;
```

WMIevent(103Fh) - WMI_P2P_INVITE_RCVD_RESULT_EVENT

Event Data:

```
00h  A_UINT16 2   oper_freq;
02h  A_UINT8  6   sa[ATH_MAC_LEN];
08h  A_UINT8  6   bssid[ATH_MAC_LEN];
0Eh  A_UINT8  1   is_bssid_valid;
0Fh  A_UINT8  6   go_dev_addr[ATH_MAC_LEN];
15h  P2P_SSID 21h ssid;
36h  A_UINT8  1   status;
```

WMIevent(1040h) - WMI_P2P_INVITE_SENT_RESULT_EVENT

Event Data:

```
00h  A_UINT8  1   status;
01h  A_UINT8  6   bssid[ATH_MAC_LEN];
07h  A_UINT8  1   is_bssid_valid;
```

WMIevent(1041h) - WMI_P2P_PROV_DISC_RESP_EVENT

Event Data:

```
00h  A_UINT8  6   peer[ATH_MAC_LEN];
06h  A_UINT16 2   config_methods;
```

WMIevent(1042h) - WMI_P2P_PROV_DISC_REQ_EVENT

Event Data:

```
00h  A_UINT8  6   sa[ATH_MAC_LEN];
06h  A_UINT16 2   wps_config_method;
08h  A_UINT8  6   dev_addr[ATH_MAC_LEN];
0Eh  A_UINT8  8   pri_dev_type[WPS_DEV_TYPE_LEN];
16h  A_UINT8  20h device_name[WPS_MAX_DEVNAME_LEN];
36h  A_UINT8  1   dev_name_len;
37h  A_UINT16 2   dev_config_methods;
39h  A_UINT8  1   device_capab;
3Ah  A_UINT8  1   group_capab;
```

WPS_DEV_TYPE_LEN = 8

WMIevent(1043h) - WMI_P2P_START_SDPD_EVENT

Event Data:

```
Unknown (none?)
```

WMIevent(1044h) - WMI_P2P_SDPD_RX_EVENT

Event Data:

```
00h  A_UINT8  1   type;
01h  A_UINT8  1   transaction_status;
02h  A_UINT8  1   dialog_token;
03h  A_UINT8  1   frag_id;
04h  A_UINT8  6   peer_addr[ATH_MAC_LEN];
0Ah  A_UINT16 2   freq;
0Ch  A_UINT16 2   status_code;
0Eh  A_UINT16 2   comeback_delay;
10h  A_UINT16 2   tlv_length;
12h  A_UINT16 2   update_indic;
14h  VAR      ..  Variable length TLV will be placed after the event
```

DSi Atheros Wifi - Unimplemented WMI WAC Functions

WMIcmd(F043h) - WMI_WAC_ENABLE_CMD aka WMI_ENABLE_WAC_CMD

Parameters:

```
00h  A_UINT32 4   period;
04h  A_UINT32 4   threshold;
08h  A_INT32  4   rssi;
0Ch  A_BOOL   4   enable;
10h  A_CHAR   8   wps_pin[8];     ;WPS related?
```

WMIcmd(F044h) - WMI_WAC_SCAN_REPLY_CMD

Parameters:

```
00h  A_ENUM   ..  cmdid ;A_UINTx or so?  (WAC_SUBCMD)
```

WAC_SUBCMD values:

```
WAC_MORE_SCAN      = -1
WAC_SEND_PROBE_IDX = 0
```

WMIcmd(F045h) - WMI_WAC_CTRL_REQ_CMD

Parameters:

```
00h  A_UINT8  1   req;     ;aka WAC_REQUEST_TYPE
01h  A_UINT8  1   cmd;     ;aka WAC_COMMAND
02h  A_UINT8  1   frame;   ;aka WAC_FRAME_TYPE
03h  A_UINT8  11h ie[17];
14h  A_INT32  4   status;  ;aka WAC_STATUS
```

WAC related constants (from wac_defs.h):

WAC_REQUEST_TYPE values:

```
WAC_SET        = Unknown (0 or 1 or so)
WAC_GET        = WAC_SET+1
```

WAC_COMMAND values:

```
WAC_ADD        = Unknown (0 or 1 or so)
WAC_DEL        = WAC_ADD+1
WAC_GET_STATUS = WAC_ADD+2
WAC_GET_IE     = WAC_ADD+3
```

WAC_FRAME_TYPE values:

```
PRBREQ         = Unknown (0 or 1 or so)
PRBRSP         = PRBREQ+1
BEACON         = PRBREQ+2
```

WAC_STATUS values:

```
WAC_FAILED_NO_WAC_AP      = -4
WAC_FAILED_LOW_RSSI       = -3
WAC_FAILED_INVALID_PARAM  = -2
WAC_FAILED_REJECTED       = -1
WAC_SUCCESS               = 0
WAC_DISABLED              = 1
```

```
  WAC_PROCEED_FIRST_PHASE   = 2
  WAC_PROCEED_SECOND_PHASE  = 3
```

WMIevent(1037h) - WMI_WAC_SCAN_DONE_EVENT
WMIevent(1038h) - WMI_WAC_REPORT_BSS_EVENT
WMIevent(1039h) - WMI_WAC_START_WPS_EVENT
WMIevent(103Ah) - WMI_WAC_CTRL_REQ_REPLY_EVENT
Event Data:
```
  Unknown (if any?)
```
WAC Events. Event data format is unknown, maybe related to below two structs:
WMI_GET_WAC_INFO structure (UNION):
```
 When some case:
  00h  A_UINT8  11h  ie[17];
 When some other case:
  00h  A_INT32  4    wac_status;
```
WMI_WPS_PIN_INFO structure: ;"WPS" might be WAC_START_WPS related?
```
  00h  A_UINT8  6  bssid[ATH_MAC_LEN];
  06h  A_UINT8  8  pin[8];        ;aka "wps_pin[8]" presumably?
```

DSi Atheros Wifi - Unimplemented WMI RF Kill and Store/Recall Functions

WMIcmd(F057h or F04Bh) - WMI_GET_RFKILL_MODE_CMD
Parameters:
```
  Unknown (none?)
```
Reply: See WMI_RFKILL_GET_MODE_CMD_EVENT

WMIcmd(F058h or F04Ch) - WMI_SET_RFKILL_MODE_CMD ;aka WMI_RFKILL_MODE_CMD
Parameters:
```
  00h  A_UINT8  1  GPIOPinNumber  ;GPIO related
  01h  A_UINT8  1  IntrType       ;?
  02h  A_UINT8  1  RadioState     ;RFKILL_RADIO_STATE
```
RFKILL_RADIO_STATE values:
```
  RADIO_STATE_OFF     = 01h
  RADIO_STATE_ON      = 02h
  RADIO_STATE_INVALID = FFh
```

WMIevent(103Bh) - WMI_RFKILL_STATE_CHANGE_EVENT
Event Data:
```
  Unknown (if any?)
```

WMIevent(103Ch) - WMI_RFKILL_GET_MODE_CMD_EVENT
Event Data:
```
  Unknown (maybe some format as in "SET_RFKILL" command parameters?)
```

WMIcmd(F05Eh or F04Eh) - WMI_STORERECALL_CONFIGURE_CMD
Parameters:
```
  00h  A_UINT8 1  enable (probably some flag)
  01h  A_UINT8 1  recipient (only one value defined: STRRCL_RECIPIENT_HOST = 1)
```
Ultra low power store/recall feature. Seems to be intended to memorize data in HOST memory... allowing the AR600x chip to power-down its on memory, or so?

WMIcmd(F05Fh or F04Fh) - WMI_STORERECALL_RECALL_CMD
Parameters:
```
  00h  A_UINT32 4  length;      ;number of bytes of data to follow
  04h  A_UINT8  .. data[1];     ;start of "RECALL" data
```

Ultra low power store/recall feature. Maybe RECALL is meant to restore data that was formerly memorized from a WMI_STORERECALL_STORE_EVENT.

WMIcmd(F060h or F050h) - WMI_STORERECALL_HOST_READY_CMD
Parameters:
```
  00h  A_UINT32 4  sleep_msec;
  04h  A_UINT8  1  store_after_tx_empty;
  05h  A_UINT8  1  store_after_fresh_beacon_rx;
```
Ultra low power store/recall feature. Whatever parameters.

WMIevent(9004h) - WMI_STORERECALL_STORE_EVENT
Event Data:
```
  00h  A_UINT32 4  msec_sleep;   ;time between power off/on
  04h  A_UINT32 4  length;       ;length of following data
  08h  A_UINT8  .. data[1];      ;start of "STORE" data
```
Ultra low power store/recall feature. Maybe this requests to memorize the "STORE" data in host memory?

DSi Atheros Wifi - Unimplemented WMI THIN Functions

WMIcmd(8000h) - WMI_THIN_RESERVED_START
WMIcmd(8FFFh) - WMI_THIN_RESERVED_END
Area for Thin commands. These command IDs can be found in "wmi_thin.h".

WMIcmd(8000h) - WMI_THIN_CONFIG_CMD
Parameters:
```
  00h  A_UINT32 4   cfgField      ;combination of WMI_THIN_CFG_...
  04h  A_UINT16 2   length        ;length in bytes of appended sub-command(s)
  06h  A_UINT8  2   reserved[2]   ;align padding
  08h  ...      ..  structure(s) selected in "cfgField"...
```
When cfgField.Bit0 set: append WMI_THIN_CONFIG_TXCOMPLETE struct:
Used to configure the params and content for TX Complete messages the will come from the Target. these messages are disabled by default but can be enabled using this structure and the WMI_THIN_CONFIG_CMDID.
```
  +00h  A_UINT8  1  version (the versioned type of messages to use, 0=disable)
  +01h  A_UINT8  1  countThreshold (msg count threshold triggering a tx
                     complete message)
  +02h  A_UINT16 2  timeThreshold (timeout interval in MSEC triggering a
                     tx complete message)
```
When cfgField.Bit1 set: append WMI_THIN_CONFIG_DECRYPT_ERR struct:
Used to configure behavior for received frames that have decryption errors. The default behavior is to discard the frame without notification. Alternately, the MAC Header is forwarded to the host with the failed status.
```
  +00h  A_UINT8  1  enable (1=send decrypt errors to the host, 0=don't)
  +01h  A_UINT8  3  reserved[3] (align padding)
```
When cfgField.Bit2 set: Unused --- NEW VERSION
```
  Unused.
```
When cfgField.Bit2 set: append WMI_THIN_CONFIG_TX_MAC_RULES --- OLD VERSION
Used to configure behavior for transmitted frames that require partial MAC header construction. These rules are used by the target to indicate which fields need to be written.
```
  +00h  A_UINT32 4  rules (combination of WMI_WRT_... values)
```
When cfgField.Bit3 set: append WMI_THIN_CONFIG_RX_FILTER_RULES struct:
Used to configure behavior for received frames as to which frames should get forwarded to the host and which should get processed internally.
```
  +00h  A_UINT32 4  rules (combination of WMI_FILT_... values)
```
When cfgField.Bit4 set: append WMI_THIN_CONFIG_CIPHER_ENCAP struct:
Used to configure behavior for both TX and RX frames regarding security cipher encapsulation. The host may specify whether or not the firmware is responsible for cipher encapsulation. If the firmware is responsible it will

add the security header and trailer for TX frames and remove the header and trailer for Rx frames. Also, the firmware will examine the resource counter if any and behave appropriately when a bad value is detected. If the host indicates responsibility for encapsulation then it is responsible for all aspects of encapsulation, however the device will still perform the encryption and decryption of the frame payload if a key has been provided.

```
  +00h  A_UINT8  1  enable (enables/disables firmware cipher encapsulation)
  +01h  A_UINT8  3  reserved[3] (align padding)
```

Summary of values for "cfgField":

```
  WMI_THIN_CFG_TXCOMP       = 00000001h
  WMI_THIN_CFG_DECRYPT      = 00000002h
  WMI_THIN_CFG_MAC_RULES    = 00000004h ;old version (or planned for future?)
  WMI_THIN_UNUSED1          = 00000004h ;current version
  WMI_THIN_CFG_FILTER_RULES = 00000008h
  WMI_THIN_CFG_CIPHER_ENCAP = 00000010h
```

WMIcmd(8001h) - WMI_THIN_SET_MIB_CMD

Parameters:

```
  00h  A_UINT16 2  length;     /* the length in bytes of the appended MIB data
  02h  A_UINT8  1  mibID;      /* the ID of the MIB element being set
  03h  A_UINT8  1  reserved;   /* align padding
```

WMIcmd(8002h) - WMI_THIN_GET_MIB_CMD ;reply?

Parameters:

```
  00h  A_UINT8  1   mibID;        /* the ID of the MIB element being set
  01h  A_UINT8  3   reserved[3];  /* align padding
```

Reply: See WMI_THIN_GET_MIB_EVENT

WMIcmd(8003h) - WMI_THIN_JOIN_CMD ;newer ver only

Parameters:

```
  00h  A_UINT32 4   basicRateMask;    /* bit mask of basic rates
  04h  A_UINT32 4   beaconIntval;     /* TUs
  08h  A_UINT16 2   atimWindow;       /* TUs
  0Ah  A_UINT16 2   channel;          /* frequency in MHz
  0Ch  A_UINT8  1   networkType;      /* INFRA_NETWORK | ADHOC_NETWORK
  0Dh  A_UINT8  1   ssidLength;       /* 0 - 32
  0Eh  A_UINT8  1   probe;            /* != 0 : issue probe req at start
  0Fh  A_UINT8  1   reserved;         /* alignment
  10h  A_UCHAR  20h ssid[WMI_MAX_SSID_LEN];
  30h  A_UINT8  6   bssid[ATH_MAC_LEN];
```

Reply: See WMI_THIN_JOIN_EVENT

WMIcmd(8004h) - WMI_THIN_CONNECT_CMD ;newer ver only

Parameters:

```
  00h  A_UINT16 2  dtim;   /* dtim interval in num beacons
  02h  A_UINT16 2  aid;    /* 80211 association ID from Assoc resp
```

WMIcmd(8005h) - WMI_THIN_RESET_CMD ;newer ver only

Parameters:

```
  00h  A_UINT8  4  reserved[4];
```

WMIevent(8000h) - WMI_THIN_EVENTID_RESERVED_START
WMIevent(8FFFh) - WMI_THIN_EVENTID_RESERVED_END

Events/Responses with special IDs for THIN stuff (wmi_thin.h)

WMIevent(8001h) - WMI_THIN_GET_MIB_EVENT

Event Data:

```
  Unknown (maybe same/similar format as for "SET_MIB" command parameters?)
```

WMIevent(8002h) - WMI_THIN_JOIN_EVENT

Event Data:

```
  00h  A_UINT8 1  result (the result of the join command)
  01h  A_UINT8 3  reserved[3]; /* alignment
```

WMI_THIN_JOIN_RESULT values (aka above "result" value):

```
  WMI_THIN_JOIN_RES_SUCCESS    = 0  ;device has joined the network
  WMI_THIN_JOIN_RES_FAIL       = 1  ;failed for unspecified reason
  WMI_THIN_JOIN_RES_TIMEOUT    = 2  ;failed due to no beacon rx in time limit
  WMI_THIN_JOIN_RES_BAD_PARAM  = 3  ;failed due to bad cmd param
  WMI_THIN_JOIN_RES_IBSS_START = 4  ;device started new IBSS network
```

MIB Access Identifiers tailored for Symbian - mibID

Below are "mibID" values for SET_MIB and GET_MIB commands, with corresponding data structures. GET/SET works only if the corresponding structure is read/write-able, as indicated by (R), (W), and (R/W). Some structures aren't actually implemented (-), ie. not read/write-able.

```
 When mibID=01h=MIB_ID_STA_MAC; WMI_THIN_MIB_STA_MAC struct:                (R)
  00h  A_UINT8  6  addr[ATH_MAC_LEN];
 When mibID=02h=MIB_ID_RX_LIFE_TIME; WMI_THIN_MIB_RX_LIFE_TIME struct:      (-)
  00h  A_UINT32 4  time (units = msec)
 When mibID=03h=MIB_ID_SLOT_TIME; WMI_THIN_MIB_SLOT_TIME struct:          (R/W)
  00h  A_UINT32 4  time (units = usec)
 When mibID=04h=MIB_ID_RTS_THRESHOLD; WMI_THIN_MIB_RTS_THRESHOLD struct:  (R/W)
  00h  A_UINT16 2  length (units = bytes)
 When mibID=05h=MIB_ID_CTS_TO_SELF; WMI_THIN_MIB_CTS_TO_SELF struct:      (R/W)
  00h  A_UINT8  1  enable (1=on, 0=off)
 When mibID=06h=MIB_ID_TEMPLATE_FRAME; WMI_THIN_MIB_TEMPLATE_FRAME struct:  (W)
  00h  A_UINT8  1  type (type of frame, 0..5, see below "FRM" values)
  01h  A_UINT8  1  rate (tx rate to be used, one of WMI_BIT_RATE)
  02h  A_UINT16 2  length (num bytes following this structure as template data)
  04h  ..       .. template data
  Frame "type" values:          frame max length:
  TEMPLATE_FRM_PROBE_REQ = 0    FRM_LEN_PROBE_REQ  = 256  ;\Symbian dictates a
  TEMPLATE_FRM_BEACON    = 1    FRM_LEN_BEACON     = 256  ; minimum of 256 for
  TEMPLATE_FRM_PROBE_RESP= 2    FRM_LEN_PROBE_RESP = 256  ;/these 3 frame types
  TEMPLATE_FRM_NULL      = 3    FRM_LEN_NULL       = 32
  TEMPLATE_FRM_QOS_NULL  = 4    FRM_LEN_QOS_NULL   = 32
  TEMPLATE_FRM_PSPOLL    = 5    FRM_LEN_PSPOLL     = 32
  Total sum of above lengths: TEMPLATE_FRM_LEN_SUM = 256+256+256+32+32+32
 When mibID=07h=MIB_ID_RXFRAME_FILTER; WMI_THIN_MIB_RXFRAME_FILTER struct:(R/W)
  00h  A_UINT32 4  filterMask;
  FRAME_FILTER_PROMISCUOUS = 00000001h
  FRAME_FILTER_BSSID       = 00000002h
 When mibID=08h=MIB_ID_BEACON_FILTER_TABLE; Several structure(s)...?        (W)
  There are three related sturctures; the actual "TABLE", and additional
  "TABLE_OUI" and "TABLE_HEADER"; unknown which of those structure(s) are
  meant to be used here...
  WMI_THIN_MIB_BEACON_FILTER_TABLE structure:
  00h  A_UINT8  1  ie;
  01h  A_UINT8  1  treatment;
  IE_FILTER_TREATMENT_CHANGE = 1
  IE_FILTER_TREATMENT_APPEAR = 2
  WMI_THIN_MIB_BEACON_FILTER_TABLE_OUI structure:
  00h  A_UINT8  1  ie;
  01h  A_UINT8  1  treatment;
  02h  A_UINT8  3  oui[3];
  05h  A_UINT8  1  type;
  06h  A_UINT16 2  version;
  WMI_THIN_MIB_BEACON_FILTER_TABLE_HEADER structure:
  00h  A_UINT16 2  numElements
  02h  A_UINT8  1  entrySize (sizeof(WMI_THIN_MIB_BEACON_FILTER_TABLE) on
  03h  A_UINT8  1  reserved                    host cpu may be 2 may be 4)
 When mibID=09h=MIB_ID_BEACON_FILTER; WMI_THIN_MIB_BEACON_FILTER struct:  (R/W)
  00h  A_UINT32 4  count (num beacons between deliveries)
  04h  A_UINT8  1  enable;
```

```
 05h  A_UINT8  3  reserved[3];
When mibID=0Ah=MIB_ID_BEACON_LOST_COUNT; WMI_THIN_MIB_BEACON_LOST_COUNT:   (W)
 00h  A_UINT32 4  count (num consec lost beacons after which send event)
When mibID=0Bh=MIB_ID_RSSI_THRESHOLD; WMI_THIN_MIB_RSSI_THRESHOLD struct:  (W)
 00h  A_UINT8 1   rssi (the low threshold which can trigger an event warning)
 01h  A_UINT8 1   tolerance (the range above and below the threshold to
                   prevent event flooding to the host)
 02h  A_UINT8 1   count (the sample count of consecutive frames necessary to
                   trigger an event)
 03h  A_UINT8 1   reserved[1] (padding)
When mibID=0Ch=MIB_ID_HT_CAP; WMI_THIN_MIB_HT_CAP struct:                  (-)
 00h  A_UINT32 4  cap;
 04h  A_UINT32 4  rxRateField;
 08h  A_UINT32 4  beamForming;
 0Ch  A_UINT8  6  addr[ATH_MAC_LEN];
 12h  A_UINT8  1  enable;
 13h  A_UINT8  1  stbc;
 14h  A_UINT8  1  maxAMPDU;
 15h  A_UINT8  1  msduSpacing;
 16h  A_UINT8  1  mcsFeedback;
 17h  A_UINT8  1  antennaSelCap;
When mibID=0Dh=MIB_ID_HT_OP; WMI_THIN_MIB_HT_OP struct:                    (-)
 00h  A_UINT32 4  infoField;
 04h  A_UINT32 4  basicRateField;
 08h  A_UINT8  1  protection;
 09h  A_UINT8  1  secondChanneloffset;
 0Ah  A_UINT8  1  channelWidth;
 0Bh  A_UINT8  1  reserved;
When mibID=0Eh=MIB_ID_HT_2ND_BEACON; WMI_THIN_MIB_HT_2ND_BEACON struct:    (-)
 00h  A_UINT8  1  cfg (see below SECOND_BEACON_xxx values)
 01h  A_UINT8  3  reserved[3] (padding)
 SECOND_BEACON_PRIMARY   = 1
 SECOND_BEACON_EITHER    = 2
 SECOND_BEACON_SECONDARY = 3
When mibID=0Fh=MIB_ID_HT_BLOCK_ACK; WMI_THIN_MIB_HT_BLOCK_ACK struct:      (-)
 00h  A_UINT8  1  txTIDField
 01h  A_UINT8  1  rxTIDField
 02h  A_UINT8  2  reserved[2] (padding)
When mibID=10h=MIB_ID_PREAMBLE; WMI_THIN_MIB_PREAMBLE struct:            (R/W)
 00h  A_UINT8  1  enableLong (1=long preamble, 0=short preamble)
 01h  A_UINT8  3  reserved[3]
When mibID=N/A=MIB_ID_GROUP_ADDR_TABLE    ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_WEP_DEFAULT_KEY_ID  ;satisfied by wmi_addKey_cmd()   (-)
When mibID=N/A=MIB_ID_TX_POWER            ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_ARP_IP_TABLE        ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_SLEEP_MODE          ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_WAKE_INTERVAL       ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_STAT_TABLE          ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_IBSS_PWR_SAVE       ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_COUNTERS_TABLE      ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_ETHERTYPE_FILTER    ;[NOT IMPLEMENTED]               (-)
When mibID=N/A=MIB_ID_BC_UDP_FILTER       ;[NOT IMPLEMENTED]               (-)
 N/A
```

THIN related constants...

MAC Header Build Rules

These values allow the host to configure the target code that is responsible for constructing the MAC header. In cases where the MAC header is provided by the host framework, the target has a diminished responsibility over what fields it must write. This will vary from framework to framework.

Symbian requires different behavior from MAC80211 which requires different behavior from MS Native Wifi.

```
 WMI_WRT_VER_TYPE  = 00000001h
 WMI_WRT_DURATION  = 00000002h
 WMI_WRT_DIRECTION = 00000004h
```

```
  WMI_WRT_POWER     = 00000008h
  WMI_WRT_WEP       = 00000010h
  WMI_WRT_MORE      = 00000020h
  WMI_WRT_BSSID     = 00000040h
  WMI_WRT_QOS       = 00000080h
  WMI_WRT_SEQNO     = 00000100h
  WMI_GUARD_TX      = 00000200h ;prevent TX ops that are not allowed for a
                                ; current state
  WMI_WRT_DEFAULT_CONFIG = 3FFh             ;<-- default all bits set
```

See also: WMI_SET_THIN_MODE_CMD

DSi Atheros Wifi - Unimplemented WMI Pyxis Functions

WMIcmd(F009h) - WMI_UNUSED1 or WMI_PYXIS_CONFIG_CMD

Parameters:

```
 Config Header:
  00h  A_UINT16 2  pyxisConfigType ;One of WMI_PYXIS_CONFIG_TYPE
  02h  A_UINT16 2  pyxisConfigLen  ;Length in Bytes of Information that follows
 When pyxisConfigType=0=WMI_PYXIS_GEN_PARAMS
  04h  A_UINT32 2  dataWindowSizeMin
  08h  A_UINT32 2  dataWindowSizeMax
  0Ch  A_UINT8  1  maxJoiners
 When pyxisConfigType=1=WMI_PYXIS_DSCVR_PARAMS
  04h  A_UINT32 4  dscvrWindow
  08h  A_UINT32 4  dscvrInterval
  0Ch  A_UINT32 4  dscvrLife
  10h  A_UINT32 4  probeInterval
  14h  A_UINT32 4  probePeriod
  18h  A_UINT16 2  dscvrChannel
 When pyxisConfigType=2=WMI_PYXIS_SET_TX_MODE
  04h  A_BOOL   4  mode
```

Whatever "Pyxis specific".

WMIcmd(F00Ah) - WMI_UNUSED2 or WMI_PYXIS_OPERATION_CMD

Parameters:

```
 Command Header:
  00h  A_UINT16 2  pyxisCmd
  02h  A_UINT16 2  pyxisCmdLen   ;Length following this header
 When pyxisCmd=0=WMI_PYXIS_DISC_PEER
  04h  A_UINT8  6  peerMacAddr[ATH_MAC_LEN]
 When pyxisCmd=1=WMI_PYXIS_JOIN_PEER
  04h  A_UINT32 4  ctrl_flags (One of the Bits determines if it
                    is Virt Adhoc/the device is to join a BSS)
  08h  A_UINT16 2  channel                  ;Data Channel
  0Ah  A_UINT8  1  networkType              ;network type
  0Bh  A_UINT8  1  dot11AuthMode            ;OPEN_AUTH
  0Ch  A_UINT8  1  authMode                 ;NONE_AUTH
  0Dh  A_UINT8  1  pairwiseCryptoType       ;One of NONE_CRYPT, AES_CRYPT
  0Eh  A_UINT8  1  pairwiseCryptoLen        ;0 since ADD_KEY passes the length
  0Fh  A_UINT8  1  groupCryptoType          ;One of NONE_CRYPT, AES_CRYPT
  10h  A_UINT8  1  groupCryptoLen           ;0 since ADD_KEY passes the length
  11h  A_UINT8  6  peerMacAddr[ATH_MAC_LEN] ;BSSID of peer network
  17h  A_UINT8  6  nwBSSID[ATH_MAC_LEN]     ;BSSID of the Pyxis Adhoc Network
 When pyxisCmd=?=WHAT? below is also "incompletely-defined" as pyxisCmd:
  04h  A_BOOL   4  mode (what is this here? dupe of WMI_PYXIS_CONFIG_CMD?)
```

Whatever "Pyxis specific".

DSi Atheros Wifi I2C EEPROM

The I2C EEPROM is read via the wifi firmware's bootstub (prior to executing the main firmware) via the SI_xxx registers. The DSi wifi boards are usually containing a HN58X2408F chip (1Kx8, with 8bit index). However, the bootstub contains code for also supporting HN58X24xx chips with bigger capacity (including such with 16bit index) (although actually using only 300h bytes regardless of the capacity). The I2C device number is A0h, or, in case of EEPROMs with 8bit index, the device number is misused to contain the upper address bits as so:

```
device = A0h + direction_flag + (addr/100h)*2  ;for devices with 8bit index
device = A0h + direction_flag                  ;for devices with 16bit index
```

I2C EEPROM Content for AR6002G

```
000h 4     Maybe Size, ID, or Version? (00000300h)
004h 2     Checksum (all halfwords at [0..2FFh] XORed shall give FFFFh)
006h 2     Unknown
008h 2     Country+8000h  ;eg. 8000h+188h=JP, 8000h+348h=US (REG_DOMAIN)
00Ah 6     MAC Address (must be same as in SPI FLASH)
010h 4     Type/version? (MSB must be 60h, verified by ARM7)
014h 4     Zerofilled
018h 5     Unknown
01Dh 1Fh   Zerofilled
03Ch 70h   FFh-filled
0ACh 8     Zerofilled
0B4h 12    Unknown
0C0h 20    Unknown
0D4h 18h   Zerofilled
0ECh 4     Unknown
0F0h 4     Unknown, overwritten by [0ECh] after loading
0F4h 12    Unknown, similar to data at 0B4h ?
100h 20    Unknown, similar to data at 0C0h ?
114h 2Ch   Zerofilled
140h 8     FFh-filled
148h 4     Unknown
14Ch 88h   Zerofilled
1D4h 3x18  Unknown
212h 18    Zerofilled
224h 4x4   Unknown       ;\
234h 2x4   Unknown       ;
23Ch 3x4   Unknown       ; together 15x4 maybe ?
248h 12    Unknown       ;
254h 3x4   Unknown       ;/
260h 60h   Unknown
2C0h 40h   Zerofilled
300h 100h  Not used (not loaded to RAM)
```

The presence of the I2C EEPROM appears to be some mis-conception. It might make sense to store the wifi calibration data on the daughterboard (rather than in eMMC storage on mainboard), however, it could be as well stored in the SPI FLASH chip, but Nintendo apparently didn't knew how to do that.

DSi Atheros Wifi Internal Hardware

Below describes the internal Atheros hardware. The hardware registers can be accessed via WINDOW_DATA, or by uploading custom code to the Xtensa CPU via BMI Bootloader commands.
Anyways, normally, the Wifi unit should be accessed via WMI commands, so one won't need to deal with internal hardware (except cases like reading the CHIP_ID, or for better understanding of the inner workings of the hardware).

Internal Xtensa CPU

Internal Memory and I/O Maps

Note: All DSi/3DS boards are using hw2 (though with minor changes between AR6002 and AR6013/AR6014 on the GPIO I2C pin wiring, and external oscillator). The hw4/hw6 variants are shown here only for curiosity, they aren't used anywhere in DSi/3DS).

AR60XX Internal Registers

DSi Atheros Wifi - Xtensa CPU Registers

Xtensa Core Registers

```
-             AR      Address registers A0..A15 (general purpose registers)
-             PC      Program Counter
```

The AR registers are used as so:

```
A0       general purpose (and return address for CALL/RET opcodes)
A1       general purpose (commonly used as stack pointer)
A2..A15  general purpose
```

Chips with Windowed Code option are actually having more than sixteen AR registers, and the register window is rotated by CALL4/CALL8/CALL12/RETW opcodes (allowing to save/restore registers without needing to

push/pop them in memory):

```
  CALL4   saves A0..A3 and moves A4..A15 to A0..A11  ;\and, probably copies
  CALL8   saves A0..A7 and moves A8..A15 to A0..A7   ; old A1 to new A1 (?),
  CALL12  saves A0..A11 and moves A12..A15 to A0..A3 ;/and new A0=ret_addr
  ENTRY   used at begin of sub-functions (allocate local variables on stack)
  RETW    windowed return (deallocate locals, and undo the CALL4/8/12 rotation)
```

The sub-functions will have retaddr/stack in A0/A1, and incoming parameters in A2 and up, and return value in A2. However, due to the rotation, the caller will see parameters/return value elsewhere (eg. for CALL8, parameters in A10 and up, and return value in A10).

The ENTRY opcode is used to allocate local variables on stack. For some weird reason ENTRY is usually also allocating "N*4 bytes" dummy space for the "N" saved words (as if they were intended to be pushed on stack, rather than being saved via rotation, maybe some exception handler is actually USING that dummy space when the register window gets full(?), the AR6002 BIOS contains a few functions that aren't allocating enough dummy words though, eg. 8EDE44h uses "entry a1,10h" although being called via CALL8).

Xtensa Special Registers

```
  0        00h   LBEG    Loop Begin                 ;\
  1        01h   LEND    Loop End                   ; Loop option
  2        02h   LCOUNT  Loop Count                 ;/
  3        03h   SAR     Shift-Amount Register      ;-Core
  4        04h   BR      Boolean Registers (16x1bit) ;-Boolean option
  5        05h   LITBASE Literal Base               ;-Literal base option
  12       0Ch   SCOMPARE1                       ;-Multiprocessor... vs S32C1I
  16       10h   ACCLO   Accumulator low (32bit)    ;\
  17       11h   ACCHI   Accumulator high (8bit)    ;
  32       20h   MR0     MAC16 register m0 (32bit)  ; MAC16 option
  33       21h   MR1     MAC16 register m1 (32bit)  ;
  34       22h   MR2     MAC16 register m2 (32bit)  ;
  35       23h   MR3     MAC16 register m3 (32bit)  ;/
  177      B1h   EPC[1]  Exception Program Counter  ;\
  232      E8h   EXCAUSE Cause of last Exception    ;
  209      D1h   EXCSAVE[1]                         ;
  230      E6h   PS                                 ; Exception option
  230      E6h   PS.EXCM                            ;
  238      EEH   EXCVADDR                           ;
  192      C0h   DEPC                               ;/
  see            PS.INTLEVEL                       ;-Interrupt option
  178..183 B2h.. EPC[2..7]                    ;\
  194..199 C2h.. EPS[2..7]                    ; High-Priority Interrupt option
  210..215 D2h.. EXCSAVE[2..7]                ;/
  234      EAh   CCOUNT                       ;\Timer Interrupt option
  240-242  F0h   CCOMPARE                     ;/
  -              AR[NAREG]                    ;\
  72       48h   WindowBase                   ; Windowed Register option
  73       49h   WindowStart                  ;
  230      E6h   PS.CALLINC                   ;
  230      E6h   PS.OWB                       ;
  230      E6h   PS.WOE                       ;/
  244-247  F4h.. MISC                         ;-Misc Special Register option
  236      ECh   ICOUNT                       ;\
  237      EDh   ICOUNTLEVEL                  ;
  128-129  80h.. IBREAKA                      ;
  96       60h   IBREAKENABLE                 ; Debug option
  144-145  90h.. DBREAKA                      ;
  160-161  A0h.. DBREAKC                      ;
  233      E9h   DEBUGCAUSE                   ;
  104      68h   DDR                          ;/
  230      E6h   PS.RING                      ;\
  83       53h   PTEVADDR                     ;
  90       5Ah   RASID                        ; MMU option
  91       5Bh   ITLBCFG                      ;
  92       5Ch   DTLBCFG                      ;
  see            ITLB                        ;
```

```
see             DTLB                          ;/
98        62h   CACHEATTR     ;-
99        63h   ATOMCTL       ;-
224       E0h   CPENABLE      ;-
226       E2h   INTERRUPT (R);\
226       E2h   INTSET    (W); Interrupt
227       E3h   INTCLEAR      ;
228       E4h   INTENABLE     ;/
106       6Ah   MEPC          ;\
107       6Bh   MEPS          ;
108       6Ch   MESAVE        ; Memory ECC/Parity
109       6Dh   MESR          ;
110       6Eh   MECR          ;
111       6Fh   MEVADDR       ;/
89        59h   MMID          ;-Trace Port
231       E7h   VECBASE       ;-
235       EBh   PRID          ;-Processor ID
```

XSR with >=64 is privileged.

Xtensa User Registers

```
0-223    0-DFh Available for designer extensions
192-255  C0h.. Reserved by Tensilica (conflicts with above "available" info?)
231      E7h   THREADPTR                      ;-Thread Pointer
232      E8h   FCR        (float control)     ;\
233      E9h   FSR        (float status)      ; Float
-              FR         (f0..f15?)          ;/
```

DSi Atheros Wifi - Xtensa CPU Core Opcodes

Core Opcodes - Move/Load/Store

```
Opcode   Native              Nocash                  Expl.
ii0st2h  L8UI   at,as,imm    movb   at,[as+imm8]     Load 8bit Unsigned
ii1st2h  L16UI  at,as,imm*2  movh   at,[as+imm8*2]   Load 16bit Unsigned
ii9st2h  L16SI  at,as,imm*2  movsh  at,[as+imm8*2]   Load 16bit Signed
ii2st2h  L32I   at,as,imm*4  mov    at,[as+imm8*4]   Load 32bit
ii4st2h  S8I    at,as,imm    movb   [as+imm8],at     Store 8bit
ii5st2h  S16I   at,as,imm*2  movh   [as+imm8*2],at   Store 16bit
ii6st2h  S32I   at,as,imm*4  mov    [as+imm8*4],at   Store 32bit
iiiit1h  L32R   at,adr       movp   at,literal       Load 32bit literal pool
iiAit2h  MOVI   at,imm12     mov    at,+/-imm12      Move Immediate(signed)
83rst0h  MOVEQZ ar,as,at     movz   at,ar,as         Move if at=0  ;zero
93rst0h  MOVNEZ ar,as,at     movnz  at,ar,as         Move if at<>0 ;nonzero
A3rst0h  MOVLTZ ar,as,at     movs   at,ar,as         Move if at<0  ;negative
B3rst0h  MOVGEZ ar,as,at     movns  at,ar,as         Move if at>=0 ;positive
```

Core Opcodes - ALU

```
Opcode   Native              Nocash                  Expl.
iiCst2h  ADDI   at,as,imm8   add  at,as,+/-imm8      Add Immediate (signed)
iiDst2h  ADDMI  at,as,imm    add  at,as,+/-imm8*256  Add Immediate*100h
80rst0h  ADD    ar,as,at     add  ar,as,at           Add        (as+at)
90rst0h  ADDX2  ar,as,at     add  ar,at,as*2         Add shift1 (as*2+at)
A0rst0h  ADDX4  ar,as,at     add  ar,at,as*4         Add shift2 (as*4+at)
B0rst0h  ADDX8  ar,as,at     add  ar,at,as*8         Add shift3 (as*8+at)
C0rst0h  SUB    ar,as,at     sub  ar,as,at           Subtract   (as-at)
D0rst0h  SUBX2  ar,as,at     sub  ar,as*2,at         Sub shift1 (as*2-at)
E0rst0h  SUBX4  ar,as,at     sub  ar,as*4,at         Sub shift2 (as*4-at)
F0rst0h  SUBX8  ar,as,at     sub  ar,as*8,at         Sub shift3 (as*8-at)
60r0t0h  NEG    ar,at        neg  ar,at              Negate
60r1t0h  ABS    ar,at        abs  ar,at              Absolute Value
10rst0h  AND    ar,as,at     and  ar,as,at           Bitwise Logical And
```

```
  20rst0h  OR     ar,as,at      or   ar,as,at  ;akaMOV  Bitwise Logical Or
  30rst0h  XOR    ar,as,at      xor  ar,as,at           Bitwise Logical Xor
```

Core Opcodes - Shift

```
  Opcode   Native               Nocash                   Expl.
  01rsi0h  SLLI   ar,as,32-imm5 shl  ar,as,32-imm5       Shift Left Logical
  21rit0h  SRAI   ar,at,imm5    sar  ar,at,imm5          Shift Right Arithmetic
  41rit0h  SRLI   ar,at,imm4    shr  ar,at,imm4          Shift Right Logical
  m4rst0h  EXTUI  ar,at,s,m     shrmask ar,at,imm5,mask  ExtractUnsignedImm
  81rst0h  SRC    ar,as,at      shr  ar,as,at,shiftreg   Shift Right Combined
  91r0t0h  SRL    ar,at         shr  ar,at,shiftreg      Shift Right Logical
  A1rs00h  SLL    ar,as         shl  ar,as,shiftreg ??   Shift Left Logical
  B1r0t0h  SRA    ar,at         sar  ar,at,shiftreg      Shift Right Arithmetic
  400s00h  SSR    as            mov  shiftreg,as         SetShiftAm for RightSh
  401s00h  SSL    as            sub  shiftreg,32,as      SetShiftAm for LeftSh
  402s00h  SSA8L  as            mov  shiftreg,as*8       SetShiftAmFor LE shift
  403s00h  SSA8B  as            sub  shiftreg,32,as*8    SetShiftAmFor BE shift
  404i.0h  SSAI   imm5          mov  shiftreg,imm5sar    SetShiftAm Immediate
```

Core Opcodes - Jump/Call

```
  Opcode   Native               Nocash                   Expl.
  iiii06h  J      adr           jmp    rel18             Unconditional Jump
  000sA0h  JX     as            jmp    as                Unconditional Jump Reg
  iiii05h  CALL0  adr           call0  rel18x4           Non-windowed Call
  000sC0h  CALLX0 as            call0  as                Non-windowed Call Reg
  000080h  RET        ;(jx a0)  ret          ;(jx a0)    Non-Windowed Return
```

Core Opcodes - Conditional Jump

```
  Opcode   Native               Nocash                   Branch if...
  iiis16h  BEQZ   as,adr        jz     as,rel12          as=0
  iiis56h  BNEZ   as,adr        jnz    as,rel12          as<>0
  iiis96h  BLTZ   as,adr        js     as,rel12          as<0         (signed)
  iiisD6h  BGEZ   as,adr        jns    as,rel12          as>=0        (signed)
  iics26h  BEQI   as,c,adr      je     as,const4,rel8    as=Imm4(c)
  iics66h  BNEI   as,c,adr      jne    as,const4,rel8    as<>Imm4(c)
  iicsA6h  BLTI   as,c,adr      jl     as,const4,rel8    as<Imm4(c)   (signed)
  iicsE6h  BGEI   as,c,adr      jge    as,const4,rel8    as>=Imm4(c)  (signed)
  iicsB6h  BLTUI  as,c,adr      jb     as,const4u,rel8   as<UnsiImm4  (unsigned)
  iicsF6h  BGEUI  as,c,adr      jae    as,const4u,rel8   as>=UnsiImm4 (unsigned)
  ii1st7h  BEQ    as,at,adr     je     as,at,rel8        as=at  equal
  ii9st7h  BNE    as,at,adr     jne    as,at,rel8        as<>at not equal
  ii2st7h  BLT    as,at,adr     jl     as,at,rel8        as<at  less  (signed)
  iiAst7h  BGE    as,at,adr     jge    as,at,rel8        as>=at gt/eq (signed)
  ii3st7h  BLTU   as,at,adr     jb     as,at,rel8        as<at  less  (unsigned)
  iiBst7h  BGEU   as,at,adr     jae    as,at,rel8        as>=at gt/eq (unsigned)
  ii0st7h  BNONE  as,at,adr     tstjz  as,at,rel8        (as AND at)=0  ;none
  ii8st7h  BANY   as,at,adr     tstjnz as,at,rel8        (as AND at)<>0 ;any set
  ii4st7h  BALL   as,at,adr     tstje  as,at,rel8        (as AND at)=at ;all set
  iiCst7h  BNALL  as,at,adr     tstjne as,at,rel8        (as AND at)<>at;not all
  ii5st7h  BBC    as,at,adr     tstjz  as,1 shl at,rel8  (as AND (1 shl at))=0
  ii6sb7h  BBCI   as,b,adr      tstjz  as,1 shl imm5,r8  (as AND (1 shl imm))=0
  iiDst7h  BBS    as,at,adr     tstjnz as,1 shl at,rel8  (as AND (1 shl at))<>0
  iiEsb7h  BBSI   as,b,adr      tstjnz as,1 shl imm5,r8  (as AND (1 shl imm))<>0
```

Core Opcodes - Misc

```
  Opcode   Native               Nocash                   Expl.
  406st0h  RER    at,as         mov  at,ext[as]          ReadExternal Register
  407st0h  WER    at,as         mov  ext[as],at          WriteExternalRegister
  03iit0h  RSR    at,imm8       mov  at,special[imm8]    ReadSpecial Register
  13iit0h  WSR    at,imm8       mov  special[imm8],at    WriteSpecialRegister
  61iit0h  XSR    at,imm8       xchg at,special[imm8]    ExchangeSpecialRegister
  002000h  ISYNC                isync                    Instruction Fetch Sync
  002010h  RSYNC                rsync                    Register Read Sync
```

```
002020h  ESYNC               esync                  Execute Synchronize
002030h  DSYNC               dsync                  Load/Store Synchronize
0020C0h  MEMW                memwait                Memory Wait
0020D0h  EXTW                extwait                External Wait
0020F0h  NOP                 nop                    No-Operation
```

Pseudo Opcodes

```
MOV    ar,as       Macro (=OR ar,as,as)
NOP                Alias for "OR An,An,An" (alternate, instead of 0020F0h)
J.L    adr,as      Macro (J or LiteralLoad+JX)
BBCI.L as,b,adr    Macro Branch Bit Clear Imm5 LE
BBSI.L as,b,adr    Macro Branch Bit Set Imm5 LE
SRLI ar,at,imm5    Alias for "SRLI ar,at,imm4" or EXTUI (when imm5>=16)
```

More (inofficial) pseudos...

```
mov    br,bs       or  br,bs,bs
mov    br,0        and br,bs,not bs
mov    br,1        or  br,bs,not bs
sub    at,as,imm   add at,as,-imm
mov    sfr_xxx     mov special[imm8]
alu    ax,...      alu ax,ax,...
```

DSi Atheros Wifi - Xtensa CPU Optional General Opcodes

Boolean Option

```
Opcode   Native              Nocash
008st0h  ANY4  bt,bs         or     bt,bs..bs+3      Any 4 Booleans True
009st0h  ALL4  bt,bs         and    bt,bs..bs+3      All 4 Booleans True
00Ast0h  ANY8  bt,bs         or     bt,bs..bs+7      Any 8 Booleans True
00Bst0h  ALL8  bt,bs         and    bt,bs..bs+7      All 8 Booleans True
02rst0h  ANDB  br,bs,bt      and    br,bs,bt         BooleanAnd
12rst0h  ANDBC br,bs,bt      and    br,bs,not bt     BooleanAndComplement(t)
22rst0h  ORB   br,bs,bt      or     br,bs,bt         BooleanOr
32rst0h  ORBC  br,bs,bt      or     br,bs,not bt     BooleanOrComplement(t)
42rst0h  XORB  br,bs,bt      xor    br,bs,bt         Boolean Xor
C3rst0h  MOVF  ar,as,bt      movz   bt,ar,as         Move if False
D3rst0h  MOVT  ar,as,bt      movnz  bt,ar,as         Move if True
ii0s76h  BF    bs,adr        jz     bs,rel8          Branch if False
ii1s76h  BT    bs,adr        jnz    bs,rel8          Branch if True
```

Misc Option

```
40Est0h  NSA    at,as        nsa    at,as            Normaliz.ShiftAmount
40Fst0h  NSAU   at,as        nsau   at,as            Norma.ShiftAmUnsigned
23rsi0h  SEXT   ar,as,imm    sext   ar,as,imm4+7     Sign Extend 7..22
33rsi0h  CLAMPS ar,as,imm    clamps ar,as,imm4+7     Signed Clamp minmax
43rst0h  MIN    ar,as,at     min    ar,as,at         Minimum Value Signed
53rst0h  MAX    ar,as,at     max    ar,as,at         Maximum Value Signed
63rst0h  MINU   ar,as,at     minu   ar,as,at         Minimum Value Unsigned
73rst0h  MAXU   ar,as,at     maxu   ar,as,at         Maximum Value Unsigned
```

Loop Option

```
ii8s76h  LOOP    as,adr      loop    as,rel8abs      Loop
ii9s76h  LOOPNEZ as,adr      loopnz  as,rel8abs      Loop if NotEqual zero
iiAs76h  LOOPGTZ as,adr      loopgtz as,rel8abs      Loop if Greater zero
```

Windowed Code Option

```
iiii15h  CALL4   adr         call4   rel18x4         Call RotateWinBy4
iiii25h  CALL8   adr         call8   rel18x4         Call RotateWinBy8
iiii35h  CALL12  adr         call12  rel18x4         Call RotateWinBy12
000sD0h  CALLX4  as          call4   as              Call RegRotateBy4
000sE0h  CALLX8  as          call8   as              Call RegRotateBy8
```

```
  000sF0h  CALLX12 as           call12  as              Call RegRotateBy12
  iiis36h  ENTRY   as,imm*8     entry   as,imm12*8      Subroutine Entry
  000090h  RETW                 retw                    Windowed-Return
  003400h  RFWO                 ret_wo                  RetFromWinOverflow
  003500h  RFWU                 ret_wu                  RetFromWinUnderflw
  001st0h  MOVSP   at,as        movsp   at,as           Move to Stack Ptr
  4080i0h  ROTW    imm4         rotw    imm4            Rotate Window -8..+7
  09ist0h  L32E    at,as,imm    mov_e   at,[as-imm*4]   Load32bitException
  49ist0h  S32E    at,as,imm    mov_e   [as-imm*4],at   StrWinForExcepts
```

Narrow Code Option

```
  --ist8h  L32I.N at,as,imm4*4  mov  at,[as+imm4*4]     Load 32bit
  --ist9h  S32I.N at,as,imm4*4  mov  [as+imm4*4],at     Store 32bit
  --rstAh  ADD.N  ar,as,at      add  ar,as,at           Add
  --rsiBh  ADDI.N ar,as,imm4    add  ar,as,imm4         Add Imm (0=-1 or 1..15)
  --is0Ch  MOVI.N as,imm        mov  as,imm7            Move Imm (-32..95)
  --is8Ch  BEQZ.N as,adr        jz   as,rel6abs         Branch if as=0
  --isCCh  BNEZ.N as,adr        jnz  as,rel6abs         Branch if as<>0
  --0stDh  MOV.N  at,as         mov  at,as              Move
  --F00Dh  RET.N     ;(jx a0)   ret        ;jx a0      Non-Windowed Return
  --F01Dh  RETW.N               retw                    Windowed Return
  --F06Dh  ILL.N                ill                     Xcept Illegal Instr.
  --Fi2Dh  BREAK.N imm4         break imm4              Debug Breakpoint
  --F03Dh  NOP.N                nop                     No-Operation
```

Mul16 Option

```
  C1rst0h  MUL16U ar,as,at      umul16  ar,as,at        Multiply16bitUnsigned
  D1rst0h  MUL16S ar,as,at      smul16  ar,as,at        Multiply16bitSigned
```

Mul32 Option

```
  82rst0h  MULL  ar,as,at       mul     ar,as,at        Multiply Low
  A2rst0h  MULUH ar,as,at       umulhi  ar,as,at        MultiplyUnsignedHigh
  B2rst0h  MULSH ar,as,at       smulhi  ar,as,at        MultiplySignedHigh
```

Div32 Option

```
  C2rst0h  QUOU ar,as,at        udiv    ar,as,at        Quotient Unsigned
  D2rst0h  QUOS ar,as,at        sdiv    ar,as,at        Quotient Signed
  E2rst0h  REMU ar,as,at        udivrem ar,as,at        Remainder Unsigned
  F2rst0h  REMS ar,as,at        sdivrem ar,as,at        Remainder Signed
```

DSi Atheros Wifi - Xtensa CPU Optional Exception/Cache/MMU Opcodes

Interrupt Option

```
  006it0h  RSIL  at,level       xchg  at,ps,intlevel i  Read/Set IntLevel
  007i00h  WAITI level          waiti ps,intlevel i     Set IntLevel and Wait
```

High-Priority Interrupt Option

```
  003i10h  RFI   level          ret_i level             RetFromHiPrioInt
```

Exception Option

```
  000000h  ILL                  ill                     Illegal Instruction
  002080h  EXCW                 xceptwait               Exception Wait
  003000h  RFE                  ret_e                   RetFromException
  003100h  RFUE                 ret_ue                  RetFromUserModeExcept
  003200h  RFDE                 ret_de                  RetFromDoubleExcept
  005000h  SYSCALL              syscall                 System Call
```

Debug Option

```
  004xy0h  BREAK imm4,imm4      break  imm8             Breakpoint
  F1E000h  RFDO                 ret_do                  RetFromDebugOperat.
  F1Es10h  RFDD    ;s=???       rer_dd imm1             RetFromDebugDispatch
```

MemECC/Parity Option

```
  003020h  RFME                 ret_me                  RetFromMemError
```

No Option

```
  E3rii0h  RUR ar,imm8          mov    ar,user[imm8]    Read User Register
  F3iit0h  WUR at,imm8          mov    user[imm8],at    WriteUserRegister
```

Region/MMU Option

```
  503st0h  RITLB0 at,as         mov    at,itlb0[as]     Read InstTLB Virtual
  507st0h  RITLB1 at,as         mov    at,itlb1[as]     Read InstTLB Translat
  50Bst0h  RDTLB0 at,as         mov    at,dtlb0[as]     Read DataTLB Virtual
  50Fst0h  RDTLB1 at,as         mov    at,dtlb1[as]     Read DataTLB Translat
  504s00h  IITLB  as            inv    itlb[as]         Invalidate InstTLB
  50Cs00h  IDTLB  as            inv    dtlb[as]         Invalidate DataTLB
  505st0h  PITLB  at,as         probe  at,itlb[as]      Probe InstTLB
  50Dst0h  PDTLB  at,as         probe  at,dtlb[as]      Probe DataTLB
  506st0h  WITLB  at,as         mov    itlb[as],at      Write InstTLB Entry
  50Est0h  WDTLB  at,as         mov    dtlb[as],at      Write DataTLB Entry
```

Multiprocessor Option

```
  iiBst2h  L32AI at,as,i*4      mov_m at,[as+imm8*4]    Load 32bit Acquire
  iiFst2h  S32RI at,as,imm*4    mov_m [as+imm8*4],at    Store 32bit Release
```

Multiprocessor Conditional Store Option

```
  iiEst2h  S32C1I at,as,imm*4   s32c1i at,[as+imm8*4]   CompareCond
```

Data Cache Option

```
  i07s82h  DPFL  as,imm4*16     cach_dpfl  [as+imm4*16] PrefetchAndLock *
  i27s82h  DHU   as,imm4*16     cach_dhu   [as+imm4*16] HitUnlock
  i37s82h  DIU   as,imm4*16     cach_diu   [as+imm4*16] Index Unlock
  i47s82h  DIWB  as,imm4*16     cach_diwb  [as+imm4*16] Index Writeback
  i57s82h  DIWBI as,imm4*16     cach_diwbi [as+imm4*16] Index WbInvali.
  ii7s02h  DPFR  as,imm8*4      cach_dpfr  [as+imm8*4]  PrefetchForRead
  ii7s12h  DPFW  as,imm8*4      cach_dpfw  [as+imm8*4]  PrefetchForWrite
  ii7s22h  DPFRO as,imm8*4      cach_dpfro [as+imm8*4]  PrefetchForRdOnce
  ii7s32h  DPFWO as,imm8*4      cach_dpfwo [as+imm8*4]  PrefetchForWrOnce
  ii7s42h  DHWB  as,imm8*4      cach_dhwb  [as+imm8*4]  HitWriteback
  ii7s52h  DHWBI as,imm8*4      cach_dhwbi [as+imm8*4]  HitWritebackInv.
  ii7s62h  DHI   as,imm8*4      cach_dhi   [as+imm8*4]  HitInvalidate
  ii7s72h  DII   as,imm8*4      cach_dii   [as+imm8*4]  Index Invalidate
```

Instruction Cache Option

```
  i07sD2h  IPFL as,imm4*16      cach_ipfl  [as+imm4*16] PrefetchAndLock *
  i27sD2h  IHU  as,imm4*16      cach_ihu   [as+imm4*16] Hit Unlock
  i37sD2h  IIU  as,imm4*16      cach_iiu   [as+imm4*16] Index Unlock
  ii7sC2h  IPF  as,imm8*4       cach_ipf   [as+imm8*4]  Prefetch
  ii7sE2h  IHI  as,imm8*4       cach_ihi   [as+imm8*4]  Hit Invalidate
  ii7sF2h  III  as,imm8*4       cach_iii   [as+imm8*4]  Index Invalidate
```

Data/Instruction Cache Test Options

```
  F18st0h  LDCT  at,as          cach_mov at,dCachTag[as]  LoadDataCacheTag
  F10st0h  LICT  at,as          cach_mov at,iCachTag[as]  LoadInstCacheTag
  F12st0h  LICW  at,as          cach_mov at,iCachDta[as]  LoadInstCacheWord
  F19st0h  SDCT  at,as          cach_mov dCachTag[as],at  StoreDataCacheTag
  F11st0h  SICT  at,as          cach_mov iCachTag[as],at  StoreInstCacheTag
  F13st0h  SICW  at,as          cach_mov iCachDta[as],at  StoreInstCacheWord
```

Unknown

```
  71xxx0h  ACCER ...            accer ...              Unknown/Unspecified
```

Custom Designer-Defined Opcode Option

```
  x6xxx0h  CUST ...             cust ...               DesignerDefinedOpcodes
```

Simcall Option

```
  005100h  SIMCALL              simcall                Non-HW Simulator-Call
```

DSi Atheros Wifi - Xtensa CPU Optional Floating-Point Opcodes

Float Option

```
  08rst0h  LSX      fr,as,at    f_mov    fr,[as+at]     LoadSingleIndexed
  ii0st3h  LSI      ft,as,imm*4 f_mov    ft,[as+imm8*4] LoadSingleImmediate
  48rst0h  SSX      fr,as,at    f_mov    [as+at],fr     Store Single Indexed
  ii4st3h  SSI      ft,as,imm*4 f_mov    [as+imm8*4],ft Store Single Immedia.
  18rst0h  LSXU     fr,as,at    f_movupd fr,[as+at]     LoadSingleIndexed+Upd
  ii8st3h  LSIU     ft,as,imm*4 f_movupd ft,[as+imm8*4] LoadSingleImm+Update
  58rst0h  SSXU     fr,as,at    f_movupd [as+at],fr     Store Single Indx+Upd
  iiCst3h  SSIU     ft,as,imm*4 f_movupd [as+imm8*4],ft Store Single Imm+Upd.
  0Arst0h  ADD.S    fr,fs,ft    f_add      fr,fs,ft     Add Single
  1Arst0h  SUB.S    fr,fs,ft    f_sub      fr,fs,ft     Subtract Single
  2Arst0h  MUL.S    fr,fs,ft    f_mul      fr,fs,ft     Multipy Single
  4Arst0h  MADD.S   fr,fs,ft    f_muladd   fr,fs,ft     Multiply+Add Single
  5Arst0h  MSUB.S   fr,fs,ft    f_mulsub   fr,fs,ft     Multiply+Sub Single
  8Arsi0h  ROUND.S  ar,fs,imm4  f_round    ar,fs,pow4   Round Single to Fixed
  9Arsi0h  TRUNC.S  ar,fs,imm4  f_trunc    ar,fs,pow4   TruncateSingleToFixed
  EArsi0h  UTRUNC.S ar,fs,imm4  f_utrunc   ar,fs,pow4   UnsignedTruncatetoFix
  AArsi0h  FLOOR.S  ar,fs,imm4  f_floor    ar,fs,pow4   FloorSingleToFixed
  BArsi0h  CEIL.S   ar,fs,imm4  f_ceil     ar,fs,pow4   Ceiling SingleToFixed
  CArsi0h  FLOAT.S  fr,as,imm4  f_float    fr,as,frac4  ConvertFixedToSingle
  DArsi0h  UFLOAT.S fr,as,imm4  f_ufloat   fr,as,frac4  UnsignedFixedToSingle
  FArs00h  MOV.S    fr,fs       f_mov      fr,fs        Move Single
  FArs10h  ABS.S    fr,fs       f_abs      fr,fs        Absolute Value Single
  FArs40h  RFR      ar,fs       f_mov      ar,fs        Move FR to AR
  FArs50h  WFR      fr,as       f_mov      fr,as        Move AR to FR
  FArs60h  NEG.S    fr,fs       f_neg      fr,fs        Negate Single
  1Brst0h  UN.S     br,fs,ft    f_cmp_un   br,fs,ft     CompareSingle Unord
  2Brst0h  OEQ.S    br,fs,ft    f_cmp_oeq  br,fs,ft     CompareSingle Equal
  3Brst0h  UEQ.S    br,fs,ft    f_cmp_ueq  br,fs,ft     CompareSingle UnordEq
  4Brst0h  OLT.S    br,fs,ft    f_cmp_olt  br,fs,ft     CompareSingle OrdLt
  5Brst0h  ULT.S    br,fs,ft    f_cmp_ult  br,fs,ft     CompareSingle UnorLt
  6Brst0h  OLE.S    br,fs,ft    f_cmp_ole  br,fs,ft     CompareSingle OrdLt/Eq
  7Brst0h  ULE.S    br,fs,ft    f_cmp_ule  br,fs,ft     CompareSingle UnorLtEq
  8Brst0h  MOVEQZ.S fr,fs,at    f_movz     at,fr,fs     Move Single if at=0
  9Brst0h  MOVNEZ.S fr,fs,at    f_movnz    at,fr,fs     Move Single if at<>0
  ABrst0h  MOVLTZ.S fr,fs,at    f_movs     at,fr,fs     Move Single if at<0
  BBrst0h  MOVGEZ.S fr,fs,at    f_movns    at,fr,fs     Move Single if at>=0
  CBrst0h  MOVF.S   fr,fs,bt    f_movz     bt,fr,fs     Move Single if bt=0
  DBrst0h  MOVT.S   fr,fs,bt    f_movnz    bt,fr,fs     Move Single if bt=1
```

pow4: (imm: opcode.bit7..4 = 0..15 aka (1 shl 0..15) aka 1..8000h)
frac4: (imm: opcode.bit7..4 = 0..15 aka (1 shr 0..15) aka 1..1/8000h)

DSi Atheros Wifi - Xtensa CPU Optional MAC16 Opcodes

MAC16 Option

```
  mw = m0..m3
```

```
mx = m0..m1
my = m2..m3
as,at = a0..a15
acc = special register acchi(8bit):acclo(32bit)
700st4h  UMUL.AA.LL as,at            umul    acc,as_l,at_l   ;\
710st4h  UMUL.AA.HL as,at            umul    acc,as_h,at_l   ; Unsigned Mul
720st4h  UMUL.AA.LH as,at            umul    acc,as_l,at_h   ; acc=as*at
730st4h  UMUL.AA.HH as,at            umul    acc,as_h,at_h   ;/
24x0y4h  MUL.DD.LL  mx,my            smul    acc,mx_l,my_l   ;\
25x0y4h  MUL.DD.HL  mx,my            smul    acc,mx_h,my_l   ; Signed Mul
26x0y4h  MUL.DD.LH  mx,my            smul    acc,mx_l,my_h   ; acc=mx*my
27x0y4h  MUL.DD.HH  mx,my            smul    acc,mx_h,my_h   ;/
340sy4h  MUL.AD.LL  as,my            smul    acc,as_l,my_l   ;\
350sy4h  MUL.AD.HL  as,my            smul    acc,as_h,my_l   ; Signed Mul
360sy4h  MUL.AD.LH  as,my            smul    acc,as_l,my_h   ; acc=as*my
370sy4h  MUL.AD.HH  as,my            smul    acc,as_h,my_h   ;/
64x0t4h  MUL.DA.LL  mx,at            smul    acc,mx_l,at_l   ;\
65x0t4h  MUL.DA.HL  mx,at            smul    acc,mx_h,at_l   ; Signed Mul
66x0t4h  MUL.DA.LH  mx,at            smul    acc,mx_l,at_h   ; acc=mx*at
67x0t4h  MUL.DA.HH  mx,at            smul    acc,mx_h,at_h   ;/
740st4h  MUL.AA.LL  as,at            smul    acc,as_l,at_l   ;\
750st4h  MUL.AA.HL  as,at            smul    acc,as_h,at_l   ; Signed Mul
760st4h  MUL.AA.LH  as,at            smul    acc,as_l,at_h   ; acc=as*at
770st4h  MUL.AA.HH  as,at            smul    acc,as_h,at_h   ;/
28x0y4h  MULA.DD.LL mx,my            smuladd acc,mx_l,my_l   ;\
29x0y4h  MULA.DD.HL mx,my            smuladd acc,mx_h,my_l   ; Signed MulAdd
2Ax0y4h  MULA.DD.LH mx,my            smuladd acc,mx_l,my_h   ; acc=acc+mx*my
2Bx0y4h  MULA.DD.HH mx,my            smuladd acc,mx_h,my_h   ;/
380sy4h  MULA.AD.LL as,my            smuladd acc,as_l,my_l   ;\
390sy4h  MULA.AD.HL as,my            smuladd acc,as_h,my_l   ; Signed MulAdd
3A0sy4h  MULA.AD.LH as,my            smuladd acc,as_l,my_h   ; acc=acc+as*my
3B0sy4h  MULA.AD.HH as,my            smuladd acc,as_h,my_h   ;/
68x0t4h  MULA.DA.LL mx,at            smuladd acc,mx_l,at_l   ;\
69x0t4h  MULA.DA.HL mx,at            smuladd acc,mx_h,at_l   ; Signed MulAdd
6Ax0t4h  MULA.DA.LH mx,at            smuladd acc,mx_l,at_h   ; acc=acc+mx*at
6Bx0t4h  MULA.DA.HH mx,at            smuladd acc,mx_h,at_h   ;/
780st4h  MULA.AA.LL as,at            smuladd acc,as_l,at_l   ;\
790st4h  MULA.AA.HL as,at            smuladd acc,as_h,at_l   ; Signed MulAdd
7A0st4h  MULA.AA.LH as,at            smuladd acc,as_l,at_h   ; acc=acc+as*at
7B0st4h  MULA.AA.HH as,at            smuladd acc,as_h,at_h   ;/
2Cx0y4h  MULS.DD.LL mx,my            smulsub acc,mx_l,my_l   ;\
2Dx0y4h  MULS.DD.HL mx,my            smulsub acc,mx_h,my_l   ; Signed MulSub
2Ex0y4h  MULS.DD.LH mx,my            smulsub acc,mx_l,my_h   ; acc=acc-mx*my
2Fx0y4h  MULS.DD.HH mx,my            smulsub acc,mx_h,my_h   ;/
3C0sy4h  MULS.AD.LL as,my            smulsub acc,as_l,my_l   ;\
3D0sy4h  MULS.AD.HL as,my            smulsub acc,as_h,my_l   ; Signed MulSub
3E0sy4h  MULS.AD.LH as,my            smulsub acc,as_l,my_h   ; acc=acc-as*my
3F0sy4h  MULS.AD.HH as,my            smulsub acc,as_h,my_h   ;/
6Cx0t4h  MULS.DA.LL mx,at            smulsub acc,mx_l,at_l   ;\
6Dx0t4h  MULS.DA.HL mx,at            smulsub acc,mx_h,at_l   ; Signed MulSub
6Ex0t4h  MULS.DA.LH mx,at            smulsub acc,mx_l,at_h   ; acc=acc-mx*at
6Fx0t4h  MULS.DA.HH mx,at            smulsub acc,mx_h,at_h   ;/
7C0st4h  MULS.AA.LL as,at            smulsub acc,as_l,at_l   ;\
7D0st4h  MULS.AA.HL as,at            smulsub acc,as_h,at_l   ; Signed MulSub
7E0st4h  MULS.AA.LH as,at            smulsub acc,as_l,at_h   ; acc=acc-as*at
7F0st4h  MULS.AA.HH as,at            smulsub acc,as_h,at_h   ;/
80ws04h  LDINC      mw,as            movupd  mw,[as+4]       ;Load+AutoInc
90ws04h  LDDEC      mw,as            movupd  mw,[as-4]       ;Load+AutoDec
```

Below opcodes are doing two separate things:

1. acc=acc+x*y ;Signed MulAdd
2. as=as+/-4, mw=[as] ;Load mw from memory (for use by NEXT opcode)

```
08wsy4h  MULA.DD.LL.LDINC mw,as,mx,my smuladd_movupd acc,mx_l,my_l,mw,[as+4]
09wsy4h  MULA.DD.HL.LDINC mw,as,mx,my smuladd_movupd acc,mx_h,my_l,mw,[as+4]
0Awsy4h  MULA.DD.LH.LDINC mw,as,mx,my smuladd_movupd acc,mx_l,my_h,mw,[as+4]
```

```
0Bwsy4h  MULA.DD.HH.LDINC mw,as,mx,my smuladd_movupd acc,mx_h,my_h,mw,[as+4]
18wsy4h  MULA.DD.LL.LDDEC mw,as,mx,my smuladd_movupd acc,mx_l,my_l,mw,[as-4]
19wsy4h  MULA.DD.HL.LDDEC mw,as,mx,my smuladd_movupd acc,mx_h,my_l,mw,[as-4]
1Awsy4h  MULA.DD.LH.LDDEC mw,as,mx,my smuladd_movupd acc,mx_l,my_h,mw,[as-4]
1Bwsy4h  MULA.DD.HH.LDDEC mw,as,mx,my smuladd_movupd acc,mx_h,my_h,mw,[as-4]
48wst4h  MULA.DA.LL.LDINC mw,as,mx,at smuladd_movupd acc,mx_l,at_l,mw,[as+4]
49wst4h  MULA.DA.HL.LDINC mw,as,mx,at smuladd_movupd acc,mx_h,at_l,mw,[as+4]
4Awst4h  MULA.DA.LH.LDINC mw,as,mx,at smuladd_movupd acc,mx_l,at_h,mw,[as+4]
4Bwst4h  MULA.DA.HH.LDINC mw,as,mx,at smuladd_movupd acc,mx_h,at_h,mw,[as+4]
58wst4h  MULA.DA.LL.LDDEC mw,as,mx,at smuladd_movupd acc,mx_l,at_l,mw,[as-4]
59wst4h  MULA.DA.HL.LDDEC mw,as,mx,at smuladd_movupd acc,mx_h,at_l,mw,[as-4]
5Awst4h  MULA.DA.LH.LDDEC mw,as,mx,at smuladd_movupd acc,mx_l,at_h,mw,[as-4]
5Bwst4h  MULA.DA.HH.LDDEC mw,as,mx,at smuladd_movupd acc,mx_h,at_h,mw,[as-4]
```

DSi Atheros Wifi - Xtensa CPU Opcode Encoding Tables

Xtensa Opcodes

Xtensa opcodes are 24bit wide (or 16bit for "narrow" opcodes), the opcodes consist of several 4bit fields (or 2bit, 8bit, 12bit, 16bit, 18bit fields in some cases):

```
23-20  19-16  15-12  11-8  7-4   3-0   Type
op2    op1    r      s     t     op0   RRR
imm4   op1    r      s     t     op0   RRI4
imm8--------> r      s     t     op0   RRI8
imm16-------------------->  t     op0   RRI16
op2    op1    rs-------->  t     op0   RSR
offset---------------------> n  op0   CALL
op2    op1    r      s     m  n  op0   CALLX
imm8--------> r      s     m  n  op0   BRI8
imm12------------->  s     m  n  op0   BRI8
               r      s     t     op0   RRRN
               imm.l  s     imm.h op0   RI7  (bit7="i")
               imm.l  s     imm.h op0   RI6  (bit7="i", bit6="z")
```

Xtensa Opcode Root- and Subtables (Summary)

Opcode decoding can be done in 4bit units, starting at the "op0" field, and then decoding further 4bit field(s) depending on the opcode. Decoding speed could be improved by grouping two 4bit fields into a 8bit field (eg. op2 and op1; that won't work out perfectly for all opcodes though).

```
ROOT\                          op0
ROOT\QRST                      op0=0, op1
ROOT\QRST\RST0                 op0=0, op1=0, op2
ROOT\QRST\RST0\ST0             op0=0, op1=0, op2=0, r
ROOT\QRST\RST0\ST0\SNM0        op0=0, op1=0, op2=0, r=0, mn
ROOT\QRST\RST0\ST0\SYNC        op0=0, op1=0, op2=0, r=2, t
ROOT\QRST\RST0\ST0\RFEI        op0=0, op1=0, op2=0, r=3, t
ROOT\QRST\RST0\ST0\RFEI\RFET op0=0, op1=0, op2=0, r=3, t=0, s
ROOT\QRST\RST0\ST1             op0=0, op1=0, op2=4, r
ROOT\QRST\RST0\TLB             op0=0, op1=0, op2=5, r
ROOT\QRST\RST0\RT0             op0=0, op1=0, op2=6, s
ROOT\QRST\RST1                 op0=0, op1=1, op2
ROOT\QRST\RST1\IMP             op0=0, op1=1, op2=F, r
ROOT\QRST\RST1\IMP\RFDX        op0=0, op1=1, op2=F, r=E, t
ROOT\QRST\RST2                 op0=0, op1=2, op2
ROOT\QRST\RST3                 op0=0, op1=3, op2
ROOT\QRST\LSCX                 op0=0, op1=8, op2
ROOT\QRST\LSC4                 op0=0, op1=9, op2
ROOT\QRST\FP0                  op0=0, op1=A, op2
ROOT\QRST\FP0\FP1OP            op0=0, op1=A, op2=F, t
ROOT\QRST\FP1                  op0=0, op1=B, op2
ROOT\LSAI                      op0=2, r
ROOT\LSAI\CACHE                op0=2, r=7, t
ROOT\LSAI\CACHE\DCE            op0=2, r=7, t=8, op1
```

```
ROOT\LSAI\CACHE\ICE          op0=2, r=7, t=D, op1
ROOT\LSCI                    op0=3, r
ROOT\MAC16                   op0=4, op2
ROOT\MAC16\MACID             op0=4, op2=0, op1
ROOT\MAC16\MACCD             op0=4, op2=1, op1
ROOT\MAC16\MACDD             op0=4, op2=2, op1
ROOT\MAC16\MACAD             op0=4, op2=3, op1
ROOT\MAC16\MACIA             op0=4, op2=4, op1
ROOT\MAC16\MACCA             op0=4, op2=5, op1
ROOT\MAC16\MACDA             op0=4, op2=6, op1
ROOT\MAC16\MACAA             op0=4, op2=7, op1
ROOT\MAC16\MACI              op0=4, op2=8, op1
ROOT\MAC16\MACC              op0=4, op2=9, op1
ROOT\CALLN                   op0=5, mn
ROOT\SI                      op0=6, mn   (and \SI\BZ, \SI\BI0, \SI\BI1)
ROOT\SI\BI1\B1               op0=6, mn=7, r
ROOT\B                       op0=7, r
ROOT\ST2                     op0=C, t
ROOT\ST3                     op0=D, r
ROOT\ST3\S3                  op0=D, r=F, t
```

Xtensa Opcode Root- and Subtables (Complete)

Below is showing the whole opcode "tree", starting with the root table (indexed via op0). Entries with "-->" are referencing to child tables, the other entries are indicating the actual opcodes (or reserved opcodes). The lower-case suffices are somewhat indicating optional opcodes.

```
   ROOT\             ROOT\QRST         ROOT\QRST\RS      ROOT\QRST\RST0\ST0
   op0               op1               op2               r
0 --> QRST           --> RST0          --> ST0           --> SNM0
1 L32R               --> RST1          AND               MOVSP
2 --> LSAI           --> RST2          OR                --> SYNC
3 --> LSCIp          --> RST3          XOR               --> RFEIx
4 --> MAC16d         EXTUI ;\          --> ST1           BREAKx
5 --> CALLN          EXTUI ;/          --> TLB           SYSCALLx
6 --> SI             CUST0 ;\          --> RT0           RSILx
7 --> B              CUST1 ;/          reserved          WAITIx (t=0)
8 L32I.Nn  ;\        --> LSCXp         ADD               ANY4p
9 S32I.Nn  ;         --> LSC4          ADDX2             ALL4p
A ADD.Nn   ; narrow  --> FP0f          ADDX4             ANY8p
B ADDI.Nn  ; 16bit   --> FP1f          ADDX8             ALL8p
C --> ST2n ;         reserved          SUB               reserved
D --> ST3n ;/        reserved          SUBX2             reserved
E reserved           reserved          SUBX4             reserved
F reserved           reserved          SUBX8             reserved

   ..\RST0\ST0\SNM0  ..\RST0\ST0\SYNC  ..\RST0\ST0\RFEI  ..\ST0\RFEI\RFET
   mn                t                 t                 s
0 ILL       ;\       ISYNC             --> RFETx         RFEx
1 reserved ; ILL     RSYNC             RFIx              RFUEx
2 reserved ;         ESYNC             RFME (s=0)        RFDEx
3 reserved ;/        DSYNC             reserved          reserved
4 reserved ;\        reserved          reserved          RFWOw
5 reserved ; N/A     reserved          reserved          RFWUw
6 reserved ;         reserved          reserved          reserved
7 reserved ;/        reserved          reserved          reserved
8 RET       ;\       EXCW              reserved          reserved
9 RETWw     ; JR     reserved          reserved          reserved
A JX        ;        reserved          reserved          reserved
B reserved ;/        reserved          reserved          reserved
C CALLX0    ;\       MEMW              reserved          reserved
D CALLX4w   ; CALLX  EXTW              reserved          reserved
E CALLX8w   ;        reserved          reserved          reserved
F CALLX12w ;/        NOP/reserved      reserved          reserved
```

	..\RST0\ST1	..\RST0\TLB	..\RST0\RT0	ROOT\QRST\RST1
	r	r	s	op2
0	SSR (t=0)	reserved	NEG	SLLI ;\
1	SSL (t=0)	reserved	ABS	SLLI ;/
2	SSA8L (t=0)	reserved	reserved	SRAI ;\
3	SSA8B (t=0)	RITLB0	reserved	SRAI ;/
4	SSAI (t=0)	IITLB (t=0)	reserved	SRLI ;-
5	reserved	PITLB	reserved	reserved
6	RER	WITLB	reserved	XSR
7	WER	RITLB1	reserved	--> ACCER (?)
8	ROTWw (s=0)	reserved	reserved	SRC
9	reserved	reserved	reserved	SRL (s=0)
A	reserved	reserved	reserved	SLL (t=0)
B	reserved	RDTLB0	reserved	SRA (s=0)
C	reserved	IDTLB (t=0)	reserved	MUL16U
D	reserved	PDTLB	reserved	MUL16S
E	NSAu	WDTLB	reserved	reserved
F	NSAUu	RDTLB1	reserved	--> IMP

	..\RST1\IMP	..\RST1\IMP\RFDX	ROOT\QRST\RST2	ROOT\QRST\RST3
	r	t	op2	op2
0	LICT	RFDO (s=0)	ABDBp	RSR
1	SICT	RFDD (s=0,1)	ANDBCp	WSR
2	LICW	reserved	ORBp	SEXTu
3	SICW	reserved	ORBCp	CLAMPSu
4	reserved	reserved	XORBp	MINu
5	reserved	reserved	reserved	MAXu
6	reserved	reserved	reserved	MINUu
7	reserved	reserved	reserved	MAXUu
8	LDCT	reserved	MULLi	MOVEQZ
9	SDCT	reserved	reserved	MOVNEZ
A	reserved	reserved	MULUHi	MOVLTZ
B	reserved	reserved	MULSHi	MOVGEZ
C	reserved	reserved	QUOUi	MOVFp
D	reserved	reserved	QUOSi	MOVTp
E	--> RFDX	reserved	REMUi	RUR
F	reserved	reserved	REMSi	WUR

	ROOT\QRST\LSCX	ROOT\QRST\LSC4	ROOT\QRST\FP0	ROOT\QRST\FP0\FP1OP
	op2	op2	op2	t
0	LSXf	L32E	ADD.Sf	MOV.Sf
1	LSXUf	reserved	SUB.Sf	ABS.Sf
2	reserved	reserved	MUL.Sf	reserved
3	reserved	reserved	reserved	reserved
4	SSXf	S32E	MADD.Sf	RFRf
5	SSXUf	reserved	MSUB.Sf	WFRf
6	reserved	reserved	reserved	NEG.Sf
7	reserved	reserved	reserved	reserved
8	reserved	reserved	ROUND.Sf	reserved
9	reserved	reserved	TRUNC.Sf	reserved
A	reserved	reserved	FLOOR.Sf	reserved
B	reserved	reserved	CEIL.Sf	reserved
C	reserved	reserved	FLOAT.Sf	reserved
D	reserved	reserved	UFLOAT.Sf	reserved
E	reserved	reserved	UTRUNC.Sf	reserved
F	reserved	reserved	--> FP1OPf	reserved

	ROOT\QRST\FP1	ROOT\LSAI	ROOT\LSAI\CACHE	..\CACHE\DCE
	op2	r	t	op1
0	reserved	L8UI	DPFRc	DPFLl
1	UN.Sf	L16UI	DPFWc	reserved
2	OEQ.Sf	L32I	DPFROc	DHUl
3	UEQ.Sf	reserved	DPFWOc	DIUl
4	OLT.Sf	S8I	DHWBc	DIWBc

5	ULT.Sf	S16I	DHWBIc	DIWBIc
6	OLE.Sf	S32I	DHIc	reserved
7	ULE.Sf	--> CACHEc	DIIc	reserved
8	MOVEQZ.Sf	reserved	--> DCEc	reserved
9	MOVNEZ.Sf	L16SI	reserved	reserved
A	MOVLTZ.Sf	MOVI	reserved	reserved
B	MOVGEZ.Sf	L32AIy	reserved	reserved
C	MOVF.Sf	ADDI	IPFc	reserved
D	MOVT.Sf	ADDMI	--> ICEc	reserved
E	reserved	S32C1Iy	IHIc	reserved
F	reserved	S32RIy	IIIc	reserved

	..\CACHE\ICE	ROOT\LSCI	ROOT\CALLN	ROOT\SI
	op1	r	mn	mn
0	IPFLl	LSIf	CALL0 ;\	J
1	reserved	reserved	CALL4 ;	BEQZ
2	IHUl	reserved	CALL8 ;	BEQI
3	IIUl	reserved	CALL12 ;/	ENTRYw
4	reserved	SSIf	CALL0 ;\	J
5	reserved	reserved	CALL4 ;	BNEZ
6	reserved	reserved	CALL8 ;	BNEI
7	reserved	reserved	CALL12 ;/	--> B1
8	reserved	LSIUf	CALL0 ;\	J
9	reserved	reserved	CALL4 ;	BLTZ
A	reserved	reserved	CALL8 ;	BLTI
B	reserved	reserved	CALL12 ;/	BLTUI
C	reserved	SSIUf	CALL0 ;\	J
D	reserved	reserved	CALL4 ;	BGEZ
E	reserved	reserved	CALL8 ;	BGEI
F	reserved	reserved	CALL12 ;/	BGEUI

	ROOT\SI\BI1\B1	ROOT\B	ROOT\ST2	ROOT\ST3
	r	r	t	r
0	BFp	BNONE	MOVI.Nn ;\	MOV.Nn
1	BTp	BEQ	MOVI.Nn ;	reserved
2	reserved	BLT	MOVI.Nn ;	reserved
3	reserved	BLTU	MOVI.Nn ;	reserved
4	reserved	BALL	MOVI.Nn ;	reserved
5	reserved	BBC	MOVI.Nn ;	reserved
6	reserved	BBCI ;\	MOVI.Nn ;	reserved
7	reserved	BBCI ;/	MOVI.Nn ;/	reserved
8	LOOP	BANY	BEQZ.Nn ;\	reserved
9	LOOPNEZ	BNE	BEQZ.Nn ;	reserved
A	LOOPGTZ	BGE	BEQZ.Nn ;	reserved
B	reserved	BGEU	BEQZ.Nn ;/	reserved
C	reserved	BNALL	BNEZ.Nn ;\	reserved
D	reserved	BBS	BNEZ.Nn ;	reserved
E	reserved	BBSI ;\	BNEZ.Nn ;	reserved
F	reserved	BBSI ;/	BNEZ.Nn ;/	--> S3

	ROOT\ST3\S3	ROOT\MAC16	ROOT\MAC16\MACI	ROOT\MAC16\MACC
	t	op2	op1	op1
0	RET.Nn	--> MACID	LDINC	LDDEC
1	RETW.Nwn	--> MACCD	reserved	reserved
2	BREAK.Nn	--> MACDD	reserved	reserved
3	NOP.Nn	--> MACAD	reserved	reserved
4	reserved	--> MACIA	reserved	reserved
5	reserved	--> MACCA	reserved	reserved
6	ILL.Nn	--> MACDA	reserved	reserved
7	reserved	--> MACAA	reserved	reserved
8	reserved	--> MACI	reserved	reserved
9	reserved	--> MACC	reserved	reserved
A	reserved	reserved	reserved	reserved
B	reserved	reserved	reserved	reserved

```
  C reserved          reserved          reserved          reserved
  D reserved          reserved          reserved          reserved
  E reserved          reserved          reserved          reserved
  F reserved          reserved          reserved          reserved

    ROOT\MAC16\MACID  ROOT\MAC16\MACCD  ROOT\MAC16\MACIA  ROOT\MAC16\MACCA
    op1               op1               op1               op1
  0 reserved          reserved          reserved          reserved
  1 reserved          reserved          reserved          reserved
  2 reserved          reserved          reserved          reserved
  3 reserved          reserved          reserved          reserved
  4 reserved          reserved          reserved          reserved
  5 reserved          reserved          reserved          reserved
  6 reserved          reserved          reserved          reserved
  7 reserved          reserved          reserved          reserved
  8 MULA.DD.LL.LDINC  MULA.DD.LL.LDDEC  MULA.DA.LL.LDINC  MULA.DA.LL.LDDEC
  9 MULA.DD.HL.LDINC  MULA.DD.HL.LDDEC  MULA.DA.HL.LDINC  MULA.DA.HL.LDDEC
  A MULA.DD.LH.LDINC  MULA.DD.LH.LDDEC  MULA.DA.LH.LDINC  MULA.DA.LH.LDDEC
  B MULA.DD.HH.LDINC  MULA.DD.HH.LDDEC  MULA.DA.HH.LDINC  MULA.DA.HH.LDDEC
  C reserved          reserved          reserved          reserved
  D reserved          reserved          reserved          reserved
  E reserved          reserved          reserved          reserved
  F reserved          reserved          reserved          reserved

    ROOT\MAC16\MACDD  ROOT\MAC16\MACAD  ROOT\MAC16\MACDA  ROOT\MAC16\MACAA
    op1               op1               op1               op1
  0 reserved          reserved          reserved          UMUL.AA.LL
  1 reserved          reserved          reserved          UMUL.AA.HL
  2 reserved          reserved          reserved          UMUL.AA.LH
  3 reserved          reserved          reserved          UMUL.AA.HH
  4 MUL.DD.LL         MUL.AD.LL         MUL.DA.LL         MUL.AA.LL
  5 MUL.DD.HL         MUL.AD.HL         MUL.DA.HL         MUL.AA.HL
  6 MUL.DD.LH         MUL.AD.LH         MUL.DA.LH         MUL.AA.LH
  7 MUL.DD.HH         MUL.AD.HH         MUL.DA.HH         MUL.AA.HH
  8 MULA.DD.LL        MULA.AD.LL        MULA.DA.LL        MULA.AA.LL
  9 MULA.DD.HL        MULA.AD.HL        MULA.DA.HL        MULA.AA.HL
  A MULA.DD.LH        MULA.AD.LH        MULA.DA.LH        MULA.AA.LH
  B MULA.DD.HH        MULA.AD.HH        MULA.DA.HH        MULA.AA.HH
  C MULS.DD.LL        MULS.AD.LL        MULS.DA.LL        MULS.AA.LL
  D MULS.DD.HL        MULS.AD.HL        MULS.DA.HL        MULS.AA.HL
  E MULS.DD.LH        MULS.AD.LH        MULS.DA.LH        MULS.AA.LH
  F MULS.DD.HH        MULS.AD.HH        MULS.DA.HH        MULS.AA.HH
```

DSi Atheros Wifi - Internal Memory Map

Overall Memory Map (Internal Xtensa memory)

Internal Memory contains I/O Ports, ROM, and RAM. Arranged as so:

```
  00000000h  I/O Ports
  000E0000h  ROM        ;\as so on AR6002 (other AR60xx chips may use
  00100000h  RAM        ;/slightly different addresses).
```

The whole memory is repeated every 400000h bytes (4MByte), this is just dumb mirroring (although Atheros is referring to it as "virtual memory"). The first three mirrors (in first 12Mbytes) are commonly used as so:

```
  00000000h..003FFFFh  used for I/O Port read/write            ;1st mirror
  00400000h..007FFFFh  used for ROM and RAM data read/write    ;2nd mirror
  00800000h..00BFFFFh  used for ROM and RAM opcode read        ;3rd mirror
  00C00000h..FFFFFFFh  normally unused                         ;4th..1024th
```

Unknown how far it is really required to use the mirrors that way. The different areas might be related to different access rights, or caching (although the AR6002/AR6003/AR6004 chips don't have any cache at all). The AR6002 BIOS does usually read ROM data from 4E0000h..4F3FFFh, however, in one case it's accidently reading ROM data from 8EEFF8h..8EF73Bh, so that's apparently possible & working, too.

Using WINDOW_DATA to read memory works well for address 00000000h..FFFFFFFFh (apart from some dangerous read-locations in the I/O area; as well as mirrors of those I/O locations).

RAM/ROM address/size, IDs, and various stuff for different chips

AR60xx chip name	AR6002	AR6003	AR6004	AR6013	AR6014
AR6002_rev alias	REV2	REV4	REV6	REV2	REV2
hw name	hw2	hw4	hw6	hw2	hw2
Nintendo DSi/3DS	Old DSi	N/A	N/A	New DSi	3DS/New3DS
Wifi Board	DWM-W015	N/A	N/A	DWM-W024	DWM-W028
SPI FLASH ID Byte	01h	N/A	N/A	02h	03h
SPI FLASH Size	4 or 128K	N/A	N/A	4K	4K
I2C EEPROM SizeUsed	300h	?	?	300h	300h
ROM Size (Kbyte)	80K	256K	512K	256K	256K
RAM Size (Kbyte)	184K	256K	256-288K?	128K	128K
IRAM Size (Kbyte)	N/A	N/A	160K?	N/A	N/A
ROM Date	23nov2007	?	?	18jul2008	25jun2009
ROM ID version	2.0.0.392	?	?	2.3.0.36	2.3.0.111
Firmware version	2.1.0.123	?	?	2.3.0.108	2.3.0.179+
ROM Base	0E0000h	0E0000h	100000h	0E0000h	0E0000h
ROM Reset Entry	8E0000h	?	?	8E0000h	8E0000h
RAM Base	100000h	140000h?	000000h??	120000h	120000h
RAM Host Interest	500400h	540600h	400600h??	520000h	520000h
RAM Start of Free	502400h	?	?	524C00h	524C00h
RAM BMI_DONE Entry	915000h	?	?	927000h	927000h
CPU Litbase	52F000h+1	?	?	54C000h+1	54C000h+1
IRAM Base	N/A	N/A	998000h	N/A	N/A
ROM Size (hex)	14000h	40000h	80000h	40000h	40000h
RAM Size (hex)	2E000h	40000h	4xxxxh?	20000h	20000h
ROM ID hex	20000188h	?	?	23000024h	2300006Fh
Firm ID hex	2100007Bh	?	?	2300006Ch	230000B3h+
CHIP_ID used	02000001h	?	?	0D000000h	0D000001h
CHIP_ID alternate?	02010001h	?	?	0D00000xh	?
BB_D2_CHIP_ID	has any?	?	?	has any?	has any?
SDIO MANFID	02000271h	?	?	02010271h	02010271h
OSC	26MHz	?	?	40MHz	40MHz

Special ROM Addresses

There are only a few ROM locations with fixed/standarized addresses:

```
Entrypoint:        ROM_Base+0
Exception Vectors: ROM_Base+xxx ?
DataSet Address:   ROM_Base+ROM_Size-8
MBIST Cksum:       ROM_Base+ROM_Size-4  ;MBIST = memory built-in-self-test ?
```

The firmware may call ROM functions, but the firmware code must be matched to specific ROM versions:
There is some call table in RAM, allowing to call (or change) function vectors, but the table's location/indices vary for different ROM versions. The RAM table doesn't contain entries for all ROM functions though, so the firmware must use hardcoded addresses (like "8E35A0h") for functions that aren't in the table.
Note that it's possible to patch ROM via TCAM/BCAM registers.

Special RAM Addresses

The first some kilobytes of RAM contain stuff like stack, call table, variables, and host interest area (but the exact addresses of that regions depend on the ROM version).
The remaining RAM could be more or less freely used by the firmware (whereas, one should probably use ROM's memory allocation function for that purpose).
The CPU's "litbase" is dictacted by the ROM, so the firmware must adapt its literal pool offsets to that value.
The default BMI_DONE entrypoint is also dictated by the ROM (although one could override it if neccessary).

Hardware I/O Addresses

The hardware I/O addresses are defined in source code folder "include\AR6002\hw*". Whereas, the folder is called "AR6002", but it does also contain defintions for AR6003 and AR6004. In a similar fashion, the source

code does also contain definitions like AR6002_REVn, whereas some of that "AR6002 revisions" are actually referring to AR6003 and AR6004. The naming scheme appears to be as so:

```
AR6002 = AR6002_REV2 = include\AR6002\hw2.0
AR6003 = AR6002_REV4 = include\AR6002\hw4.0
AR6004 = AR6002_REV6 = include\AR6002\hw6.0
```

The above folders are included in all source code versions (newer atheros code from 2010 is definetly no longer compatible with the DSi's AR6002 firmware, however, concerning the hardware definitions, the "hw2.0" folder appears to have stayed intact and should be still compatible with real hardware, even in newer source code versions).
However, there are at least two different "hw4.0" versions, the newer one containing some additional registers, and some changed register addresses. There's also some AR6002_REV42 definition in some files, which might be related to that "hw4.0" variant. Current theory would be that "hw4.0" means AR6003, and the newer "hw4.0" should be actually "hw4.2" and means some different chip with unknown name. Or whatever.
There's also a "include\AR6002\hw" folder, this is just some useless dupe, containing the exact same files as "include\AR6002\hw2.0".
There's also something called "MCKINLEY", which seems to same (or similar) as "AR6004".
The actual files in the above folders are containing some very ugly bloated auto-generated definitions, definetly not suitable for human reading (except by using some software parser for extracting relevant definitions).

DSi Atheros Wifi - Internal I/O Map Summary (hw2)

Overall Summary (hw2.0)

```
004000h 1C4h    Clock/RTC Registers (rtc_reg.h) (hw2.0)
008000h 208h    Memory Controller (TCAM) (vmc_reg.h) (hw2.0)
00C000h 40h     Serial UART (uart_reg.h) (hw2.0)
010000h 18h     Serial I2C/SPI Interface (si_reg.h) (hw2.0)
0140F4h 4       GPIO Registers (gpio_reg.h) (hw2.0)
018000h 114h    MBOX Registers (mbox_reg.h) (hw2.0)
01A000h 2000h   HOST_IF_WINDOW (mbox_reg.h) (hw2.0)
01C000h 64h     Analog Intf Registers (analog_reg.h) (hw2.0)
01C080h 10h     Analog Intf Registers (analog_intf_reg.h) (hw2.0)
020000h ?       MAC DMA maybe, if any ?
028000h 1800h   MAC PCU Registers (mac_pcu.h) (hw2.0)
029800h ...     BB/LC maybe...
029800h ?       Used in hw2, but undocumented... maybe BASEBAND_0 alike hw4?
02A000h ?       Used in hw2, but undocumented... maybe BASEBAND_1 alike hw4?
030000h ?       Used in hw2, but undocumented... maybe RDMA or EFUSE or so?
```

Clock/RTC Registers (rtc_reg.h) (hw2.0)

```
004000h 4       (WLAN_)RESET_CONTROL
004004h 4       (WLAN_)XTAL_CONTROL
004008h 4       (WLAN_)TCXO_DETECT
00400Ch 4       (WLAN_)XTAL_TEST
004010h 4       (WLAN_)QUADRATURE
004014h 4       (WLAN_)PLL_CONTROL
004018h 4       (WLAN_)PLL_SETTLE
00401Ch 4       (WLAN_)XTAL_SETTLE
004020h 4       (WLAN_)CPU_CLOCK
004024h 4       (WLAN_)CLOCK_OUT
004028h 4       (WLAN_)CLOCK_CONTROL
00402Ch 4       (WLAN_)BIAS_OVERRIDE
004030h 4       (WLAN_)WDT_CONTROL          ;\
004034h 4       (WLAN_)WDT_STATUS           ;
004038h 4       (WLAN_)WDT                  ; Watchdog Timer
00403Ch 4       (WLAN_)WDT_COUNT            ;
004040h 4       (WLAN_)WDT_RESET            ;/
004044h 4       (WLAN_)INT_STATUS           ;-Interrupt Status
004048h 4       (WLAN_)LF_TIMER0            ;\
00404Ch 4       (WLAN_)LF_TIMER_COUNT0      ; Low-Freq Timer
```

```
004050h 4       (WLAN_)LF_TIMER_CONTROL0      ;
004054h 4       (WLAN_)LF_TIMER_STATUS0       ;/
004058h 4       (WLAN_)LF_TIMER1              ;\
00405Ch 4       (WLAN_)LF_TIMER_COUNT1        ; Low-Freq Timer
004060h 4       (WLAN_)LF_TIMER_CONTROL1      ;
004064h 4       (WLAN_)LF_TIMER_STATUS1       ;/
004068h 4       (WLAN_)LF_TIMER2              ;\
00406Ch 4       (WLAN_)LF_TIMER_COUNT2        ; Low-Freq Timer
004070h 4       (WLAN_)LF_TIMER_CONTROL2      ;
004074h 4       (WLAN_)LF_TIMER_STATUS2       ;/
004078h 4       (WLAN_)LF_TIMER3              ;\
00407Ch 4       (WLAN_)LF_TIMER_COUNT3        ; Low-Freq Timer
004080h 4       (WLAN_)LF_TIMER_CONTROL3      ;
004084h 4       (WLAN_)LF_TIMER_STATUS3       ;/
004088h 4       (WLAN_)HF_TIMER               ;\
00408Ch 4       (WLAN_)HF_TIMER_COUNT         ; High-Freq Timer
004090h 4       (WLAN_)HF_LF_COUNT      ;<--  ;
004094h 4       (WLAN_)HF_TIMER_CONTROL       ;
004098h 4       (WLAN_)HF_TIMER_STATUS        ;/
00409Ch 4       (WLAN_)RTC_CONTROL            ;\
0040A0h 4       (WLAN_)RTC_TIME               ;
0040A4h 4       (WLAN_)RTC_DATE               ;
0040A8h 4       (WLAN_)RTC_SET_TIME           ; Real-Time Clock
0040ACh 4       (WLAN_)RTC_SET_DATE           ;
0040B0h 4       (WLAN_)RTC_SET_ALARM          ;
0040B4h 4       (WLAN_)RTC_CONFIG             ;
0040B8h 4       (WLAN_)RTC_ALARM_STATUS       ;/
0040BCh 4       (WLAN_)UART_WAKEUP
0040C0h 4       (WLAN_)RESET_CAUSE
0040C4h 4       (WLAN_)SYSTEM_SLEEP
0040C8h 4       (WLAN_)SDIO_WRAPPER
0040CCh 4       (WLAN_)MAC_SLEEP_CONTROL
0040D0h 4       (WLAN_)KEEP_AWAKE
0040D4h 4       (WLAN_)LPO_CAL_TIME                 ;\
0040D8h 4       (WLAN_)LPO_INIT_DIVIDEND_INT        ;
0040DCh 4       (WLAN_)LPO_INIT_DIVIDEND_FRACTION   ;
0040E0h 4       (WLAN_)LPO_CAL                      ;
0040E4h 4       (WLAN_)LPO_CAL_TEST_CONTROL         ;
0040E8h 4       (WLAN_)LPO_CAL_TEST_STATUS          ;/
0040ECh 4       (WLAN_)CHIP_ID
0040F0h 4       (WLAN_)DERIVED_RTC_CLK
0040F4h 4       MAC_PCU_SLP32_MODE
0040F8h 4       MAC_PCU_SLP32_WAKE
0040FCh 4       MAC_PCU_SLP32_INC
004100h 4       MAC_PCU_SLP_MIB1
004104h 4       MAC_PCU_SLP_MIB2
004108h 4       MAC_PCU_SLP_MIB3
00410Ch 4       MAC_PCU_SLP_BEACON          ;<-- hw2.0 only (not hw4.0)
004110h 4       (WLAN_)POWER_REG            ;\located here in hw2.0
004114h 4       (WLAN_)CORE_CLK_CTRL        ;/
004118h 1x8     PAD0                        ;\
004120h 4x8     SDIO_SETUP_CIRCUIT[8]       ;
004140h 4       SDIO_SETUP_CONFIG           ;
004144h 4       CPU_SETUP_CONFIG            ; hw2.0 only (not hw4.0)
004148h 1x24    PAD1                        ;
004160h 4x8     CPU_SETUP_CIRCUIT[8]        ;
004180h 4       BB_SETUP_CONFIG             ;
004184h 1x28    PAD2                        ;
0041A0h 4x8     BB_SETUP_CIRCUIT[8]         ;/
0041C0h 4       (WLAN_)GPIO_WAKEUP_CONTROL  ;-located here in hw2.0
```

Memory Controller (vmc_reg.h) (hw2.0)

```
008000h 4x32    (WLAN_)MC_TCAM_VALID[0..31]         ;\
008080h 4x32    (WLAN_)MC_TCAM_MASK[0..31]          ; ROM Patches
```

```
008100h 4x32    (WLAN_)MC_TCAM_COMPARE[0..31]       ;
008180h 4x32    (WLAN_)MC_TCAM_TARGET[0..31]        ;/
008200h 4       (WLAN_)ADDR_ERROR_CONTROL           ;\ADDR_ERROR
008204h 4       (WLAN_)ADDR_ERROR_STATUS            ;/
```

Serial UART (uart_reg.h) (hw2.0)

```
00C000h 4       (WLAN_UART_)RBR  - RX Data FIFO (R)           (when DLAB=0)
00C000h 4       (WLAN_UART_)THR  - TX Data FIFO (W)           (when DLAB=0)
00C000h 4       (WLAN_UART_)DLL  - Baudrate Divisor LSB (R/W) (when DLAB=1)
00C004h 4       (WLAN_UART_)IER  - Interrupt Control (R/W)    (when DLAB=0)
00C004h 4       (WLAN_UART_)DLH  - Baudrate Divisor MSB (R/W) (when DLAB=1)
00C008h 4       (WLAN_UART_)IIR  - Interrupt Status (R)
00C008h 4       (WLAN_UART_)FCR  - FIFO Control (W)
00C00Ch 4       (WLAN_UART_)LCR  - Character Format Control (R/W)
00C010h 4       (WLAN_UART_)MCR  - Handshaking Control (R/W)
00C014h 4       (WLAN_UART_)LSR  - RX/TX Status (R) (W=don't do)
00C018h 4       (WLAN_UART_)MSR  - Handshaking Status (R) (W=don't do)
00C01Ch 4       (WLAN_UART_)SCR  - Scratch (R/W)
00C020h 4       (WLAN_UART_)SRBR - (mirror of RBR?)          (when DLAB=0?)
00C024h 1x4     PAD0
00C028h 4       (WLAN_UART_)SIIR - (mirror or IIR?)
00C02Ch 4       (WLAN_UART_?)MWR - Whatever "M Write Register?"
00C030h 1x4     PAD1
00C034h 4       (WLAN_UART_)SLSR - (mirror or LSR?) <-- used by AR6002 ROM
00C038h 4       (WLAN_UART_)SMSR - (mirror of MSR?)
00C03Ch 4       (WLAN_UART_?)MRR - Whatever "M Read Register?"
```

Serial I2C/SPI Interface (si_reg.h) (hw2.0)

```
010000h 4       SI_CONFIG
010004h 4       SI_CS
010008h 4       SI_TX_DATA0
01000Ch 4       SI_TX_DATA1
010010h 4       SI_RX_DATA0
010014h 4       SI_RX_DATA1
```

GPIO Registers (gpio_reg.h) (hw2.0)

```
014000h 4       (WLAN_)GPIO_OUT          ;\GPIO Output Data
014004h 4       (WLAN_)GPIO_OUT_W1TS     ; (direct, and Write-1-To-Set/Clr)
014008h 4       (WLAN_)GPIO_OUT_W1TC     ;/
01400Ch 4       (WLAN_)GPIO_ENABLE       ;\GPIO Output Enable
014010h 4       (WLAN_)GPIO_ENABLE_W1TS  ; (direct, and Write-1-To-Set/Clr)
014014h 4       (WLAN_)GPIO_ENABLE_W1TC  ;/
014018h 4       (WLAN_)GPIO_IN           ;-GPIO Input
01401Ch 4       (WLAN_)GPIO_STATUS       ;\GPIO Interrupt Status
014020h 4       (WLAN_)GPIO_STATUS_W1TS  ; (direct, and Write-1-To-Set/Clr)
014024h 4       (WLAN_)GPIO_STATUS_W1TC  ;/
014028h 4       (WLAN_)GPIO_PIN0   ;GPIO0  Bluetooth coex BT_PRIORITY
01402Ch 4       (WLAN_)GPIO_PIN1   ;GPIO1  Bluetooth coex WLAN_ACTIVE
014030h 4       (WLAN_)GPIO_PIN2   ;GPIO2  Bluetooth coex BT_FREQ    ;I2C SCL
014034h 4       (WLAN_)GPIO_PIN3   ;GPIO3  Bluetooth coex BT_ACTIVE  ;I2C SDA
014038h 4       (WLAN_)GPIO_PIN4   ;GPIO4  SDIO/GSPI interface select
01403Ch 4       (WLAN_)GPIO_PIN5   ;GPIO5  SDIO/GSPI interface select
014040h 4       (WLAN_)GPIO_PIN6   ;GPIO6  -
014044h 4       (WLAN_)GPIO_PIN7   ;GPIO7  TRST for JTAG debug
014048h 4       (WLAN_)GPIO_PIN8   ;GPIO8  external 32kHz clock in
01404Ch 4       (WLAN_)GPIO_PIN9   ;GPIO9  I2C SCL  or SPI CLK
014050h 4       (WLAN_)GPIO_PIN10  ;GPIO10 I2C SDA  or SPI MISO
014054h 4       (WLAN_)GPIO_PIN11  ;GPIO11 UART RXD or SPI MOSI
014058h 4       (WLAN_)GPIO_PIN12  ;GPIO12 UART TXD or SPI /CS
01405Ch 4       (WLAN_)GPIO_PIN13  ;GPIO13 Reset in for JTAG debug
014060h 4       (WLAN_)GPIO_PIN14  ;GPIO14 UART CTS
014064h 4       (WLAN_)GPIO_PIN15  ;GPIO15 UART RTS
014068h 4       (WLAN_)GPIO_PIN16  ;GPIO16 -
01406Ch 4       (WLAN_)GPIO_PIN17  ;GPIO17 -
```

```
014070h 4       SDIO_PIN    - Config: Pad Pull/Strength
014074h 4       CLK_REQ_PIN - Config: Pad Pull/Strength/AteOeLow
014078h 4       (WLAN_)SIGMA_DELTA
01407Ch 4       (WLAN_)DEBUG_CONTROL
014080h 4       (WLAN_)DEBUG_INPUT_SEL
014084h 4       (WLAN_)DEBUG_OUT
014088h 4       LA_CONTROL
01408Ch 4       LA_CLOCK
014090h 4       LA_STATUS
014094h 4       LA_TRIGGER_SAMPLE
014098h 4       LA_TRIGGER_POSITION
01409Ch 4       LA_PRE_TRIGGER
0140A0h 4       LA_POST_TRIGGER
0140A4h 4       LA_FILTER_CONTROL
0140A8h 4       LA_FILTER_DATA
0140ACh 4       LA_FILTER_WILDCARD
0140B0h 4       LA_TRIGGERA_DATA
0140B4h 4       LA_TRIGGERA_WILDCARD
0140B8h 4       LA_TRIGGERB_DATA
0140BCh 4       LA_TRIGGERB_WILDCARD
0140C0h 4       LA_TRIGGER
0140C4h 4       LA_FIFO
0140C8h 4x2     LA[0..1]
0140D0h 4       ANT_PIN     - Config: Pad Pull/Strength
0140D4h 4       ANTD_PIN    - Config: Pad Pull
0140D8h 4       GPIO_PIN    - Config: Pad Pull/Strength
0140DCh 4       GPIO_H_PIN  - Config: Pad Pull
0140E0h 4       BT_PIN      - Config: Pad Pull/Strength
0140E4h 4       BT_WLAN_PIN - Config: Pad Pull
0140E8h 4       SI_UART_PIN - Config: Pad Pull/Strength
0140ECh 4       CLK32K_PIN  - Config: Pad Pull
0140F0h 4       (WLAN_)RESET_TUPLE_STATUS
```

MBOX Registers (mbox_reg.h) (hw2.0)

```
018000h 4x4     (WLAN_)MBOX_FIFO[0..3]
018010h 4       (WLAN_)MBOX_FIFO_STATUS
018014h 4       (WLAN_)MBOX_DMA_POLICY
018018h 4       (WLAN_)MBOX0_DMA_RX_DESCRIPTOR_BASE   ;\
01801Ch 4       (WLAN_)MBOX0_DMA_RX_CONTROL           ; MBOX 0
018020h 4       (WLAN_)MBOX0_DMA_TX_DESCRIPTOR_BASE   ;
018024h 4       (WLAN_)MBOX0_DMA_TX_CONTROL           ;/
018028h 4       (WLAN_)MBOX1_DMA_RX_DESCRIPTOR_BASE   ;\
01802Ch 4       (WLAN_)MBOX1_DMA_RX_CONTROL           ; MBOX 1
018030h 4       (WLAN_)MBOX1_DMA_TX_DESCRIPTOR_BASE   ;
018034h 4       (WLAN_)MBOX1_DMA_TX_CONTROL           ;/
018038h 4       (WLAN_)MBOX2_DMA_RX_DESCRIPTOR_BASE   ;\
01803Ch 4       (WLAN_)MBOX2_DMA_RX_CONTROL           ; MBOX 2
018040h 4       (WLAN_)MBOX2_DMA_TX_DESCRIPTOR_BASE   ;
018044h 4       (WLAN_)MBOX2_DMA_TX_CONTROL           ;/
018048h 4       (WLAN_)MBOX3_DMA_RX_DESCRIPTOR_BASE   ;\
01804Ch 4       (WLAN_)MBOX3_DMA_RX_CONTROL           ; MBOX 3
018050h 4       (WLAN_)MBOX3_DMA_TX_DESCRIPTOR_BASE   ;
018054h 4       (WLAN_)MBOX3_DMA_TX_CONTROL           ;/
018058h 4       (WLAN_)MBOX_INT_STATUS                ;\Interrupt
01805Ch 4       (WLAN_)MBOX_INT_ENABLE                ;/
018060h 4       (WLAN_)INT_HOST
018064h 1x28    PAD0
018080h 4x8     (WLAN_)LOCAL_COUNT[0..7]
0180A0h 4x8     (WLAN_)COUNT_INC[0..7]
0180C0h 4x8     (WLAN_)LOCAL_SCRATCH[0..7]
0180E0h 4       (WLAN_)USE_LOCAL_BUS
0180E4h 4       (WLAN_)SDIO_CONFIG
0180E8h 4       (WLAN_)MBOX_DEBUG
0180ECh 4       (WLAN_)MBOX_FIFO_RESET
```

```
0180F0h 4x4     (WLAN_)MBOX_TXFIFO_POP[0..3]
018100h 4x4     (WLAN_)MBOX_RXFIFO_POP[0..3]
018110h 4       (WLAN_)SDIO_DEBUG
018114h 1x7916  PAD1
01A000h 4x2048  (WLAN_)HOST_IF_WINDOW[0..2047]
```

Analog Intf Registers (analog_reg.h) (hw2.0)

```
01C000h 4       SYNTH_SYNTH1        ;\
01C004h 4       SYNTH_SYNTH2        ;
01C008h 4       SYNTH_SYNTH3        ;
01C00Ch 4       SYNTH_SYNTH4        ; also defined in "synth_reg.h"
01C010h 4       SYNTH_SYNTH5        ;
01C014h 4       SYNTH_SYNTH6        ;
01C018h 4       SYNTH_SYNTH7        ;
01C01Ch 4       SYNTH_SYNTH8        ;/
01C020h 4       RF5G_RF5G1          ;\also defined in "rf5G_reg.h"
01C024h 4       RF5G_RF5G2          ;/
01C028h 4       RF2G_RF2G1          ;\also defined in "rf2G_reg.h"
01C02Ch 4       RF2G_RF2G2          ;/
01C030h 4       TOP_GAIN            ;\also defined in "top_reg.h"
01C034h 4       TOP_TOP             ;/
01C038h 4       BIAS_BIAS_SEL       ;\
01C03Ch 4       BIAS_BIAS1          ; also defined in "bias_reg.h"
01C040h 4       BIAS_BIAS2          ;
01C044h 4       BIAS_BIAS3          ;/
01C048h 4       TXPC_TXPC           ;\also defined in "txpc_reg.h"
01C04Ch 4       TXPC_MISC           ;/
01C050h 4       RXTXBB_RXTXBB1      ;\
01C054h 4       RXTXBB_RXTXBB2      ; also defined in "rxtxbb_reg.h"
01C058h 4       RXTXBB_RXTXBB3      ;
01C05Ch 4       RXTXBB_RXTXBB4      ;/
01C060h 4       ADDAC_ADDAC1        ;-also defined in "addac.h"
01C064h 1x1Ch   -
```

More Analog Intf Registers (analog_intf_reg.h) (hw2.0)

```
01C080h 4       SW_OVERRIDE         ;\
01C084h 4       SIN_VAL             ; defined ONLY in "analog_intf_reg.h"
01C088h 4       SW_SCLK             ;
01C08Ch 4       SW_CNTL             ;/
```

MAC PCU Registers (mac_pcu.h) (hw2.0)

```
028000h (00h) - REG_STA_ID0             ;aka MAC_PCU_STA_ADDR_L32
028004h (01h) - REG_STA_ID1             ;aka MAC_PCU_STA_ADDR_U16
028008h (02h) - REG_BSS_ID0             ;aka MAC_PCU_BSSID_L32
02800Ch (03h) - REG_BSS_ID1             ;aka MAC_PCU_BSSID_U16
028010h (04h) - MAC_PCU_REG_BCNRSSI     ;aka MAC_PCU_BCN_RSSI_AVE
028014h (05h) - REG_TIME_OUT            ;aka MAC_PCU_ACK_CTS_TIMEOUT
028018h (06h) - MAC_PCU_REG_BCNSIG      ;aka MAC_PCU_BCN_RSSI_CTL
02801Ch (07h) - REG_USEC                ;aka MAC_PCU_USEC_LATENCY
028020h (08h) - REG_BEACON
028024h (09h) - REG_CFP_PERIOD          ;aka (MAC_???)PCU_MAX_CFP_DUR (?)
028028h (0Ah) - REG_TIMER0
02802Ch (0Bh) - REG_TIMER1
028030h (0Ch) - REG_TIMER2
028034h (0Dh) - REG_TIMER3
028038h (0Eh) - REG_CFP_DUR             ;aka (MAC_???)PCU_MAX_CFP_DUR (?)
02803Ch (0Fh) - REG_RX_FILTER           ;aka MAC_PCU_RX_FILTER
028040h (10h) - REG_MCAST_FIL0          ;aka MAC_PCU_MCAST_FILTER_L32
028044h (11h) - REG_MCAST_FIL1          ;aka MAC_PCU_MCAST_FILTER_U32
028048h (12h) - MAC_PCU_REG_DIAGSW      ;aka MAC_PCU_DIAG_SW
02804Ch (13h) - REG_TSF_L32
028050h (14h) - REG_TSF_U32
028054h (15h) - REG_TST_ADDAC           ;aka MAC_PCU_TST_ADDAC
028058h (16h) - REG_DEF_ANT             ;aka MAC_PCU_DEF_ANTENNA
```

```
02805Ch (17h) - MAC_PCU_REG_MUTE_MASKS0 ;aka MAC_PCU_AES_MUTE_MASK_0
028060h (18h) - MAC_PCU_REG_MUTE_MASKS1 ;aka MAC_PCU_AES_MUTE_MASK_1
028064h (19h) - MAC_PCU_REG_GATED_CLKS  ;aka MAC_PCU_GATED_CLKS
028068h (1Ah) - MAC_PCU_REG_OBS2        ;aka MAC_PCU_OBS_BUS_2
02806Ch (1Bh) - MAC_PCU_REG_OBS1        ;aka MAC_PCU_OBS_BUS_1
028070h (1Ch..1Fh) - N/A
028080h (20h) - REG_LAST_TSTP           ;aka MAC_PCU_LAST_BEACON_TSF (?)
028084h (21h) - REG_NAV                 ;aka MAC_PCU_NAV
028088h (22h) - REG_RTS_OK              ;aka MAC_PCU_RTS_SUCCESS_CNT
02808Ch (23h) - REG_RTS_FAIL            ;aka MAC_PCU_RTS_FAIL_CNT
028090h (24h) - REG_ACK_FAIL            ;aka MAC_PCU_ACK_FAIL_CNT
028094h (25h) - REG_FCS_FAIL            ;aka MAC_PCU_FCS_FAIL_CNT
028098h (26h) - REG_BEACON_CNT          ;aka MAC_PCU_BEACON_CNT
02809Ch (27h..2Fh) - N/A
0280C0h (30h) - MAC_PCU_REG_XRMODE      ;aka MAC_PCU_XRMODE
0280C4h (31h) - MAC_PCU_REG_XRDEL       ;aka MAC_PCU_XRDEL
0280C8h (32h) - MAC_PCU_REG_XRTO        ;aka MAC_PCU_XRTO
0280CCh (33h) - MAC_PCU_REG_XRCRP       ;aka MAC_PCU_XRCRP
0280D0h (34h) - MAC_PCU_REG_XRSTMP      ;aka MAC_PCU_XRSTMP
0280D4h (35h) - MAC_PCU_REG_SLP1        ;aka MAC_PCU_SLP1 ;\moved to
0280D8h (36h) - MAC_PCU_REG_SLP2        ;aka MAC_PCU_SLP2 ; 004xxxh/005xxxh
0280DCh (37h) - (//MAC_PCU_REG_SLP3)    ;aka MAC_PCU_SLP3 ;/in hw4/hw6 (!)
0280E0h (38h) - MAC_PCU_REG_BSSMSKL     ;aka MAC_PCU_ADDR1_MASK_L32
0280E4h (39h) - MAC_PCU_REG_BSSMSKH     ;aka MAC_PCU_ADDR1_MASK_U16
0280E8h (3Ah) - MAC_PCU_REG_TPC         ;aka MAC_PCU_TPC
0280ECh (3Bh) - MAC_PCU_REG_TFC         ;aka MAC_PCU_TX_FRAME_CNT
0280F0h (3Ch) - MAC_PCU_REG_RFC         ;aka MAC_PCU_RX_FRAME_CNT
0280F4h (3Dh) - MAC_PCU_REG_RCC         ;aka MAC_PCU_RX_CLEAR_CNT
0280F8h (3Eh) - MAC_PCU_REG_CC          ;aka MAC_PCU_CYCLE_CNT
0280FCh (3Fh) - MAC_PCU_REG_QT1         ;aka MAC_PCU_QUIET_TIME_1
028100h (40h) - MAC_PCU_REG_QT2         ;aka MAC_PCU_QUIET_TIME_2
028104h (41h) - MAC_PCU_REG_TSF
028108h (42h) - MAC_PCU_REG_NOACK       ;aka MAC_PCU_QOS_NO_ACK
02810Ch (43h) - MAC_PCU_REG_PHYERR      ;aka MAC_PCU_PHY_ERROR_MASK
028110h (44h) - MAC_PCU_REG_XRLAT       ;aka MAC_PCU_XRLAT
028114h (45h) - MAC_PCU_REG_ACKSIFS_RESERVED
028118h (46h) - MAC_PCU_REG_MICQOSCTL   ;aka MAC_PCU_MIC_QOS_CONTROL
02811Ch (47h) - MAC_PCU_REG_MICQOSSEL   ;aka MAC_PCU_MIC_QOS_SELECT
028120h (48h) - MAC_PCU_REG_MISCMODE    ;aka MAC_PCU_MISC_MODE
028124h (49h) - MAC_PCU_REG_FILTOFDM    ;aka MAC_PCU_FILTER_OFDM_CNT
028128h (4Ah) - MAC_PCU_REG_FILTCCK     ;aka MAC_PCU_FILTER_CCK_CNT
02812Ch (4Bh) - MAC_PCU_REG_PHYCNT1     ;aka MAC_PCU_PHY_ERR_CNT_1
028130h (4Ch) - MAC_PCU_REG_PHYCNTMASK1 ;aka MAC_PCU_PHY_ERR_CNT_1_MASK
028134h (4Dh) - MAC_PCU_REG_PHYCNT2     ;aka MAC_PCU_PHY_ERR_CNT_2
028138h (4Eh) - MAC_PCU_REG_PHYCNTMASK2 ;aka MAC_PCU_PHY_ERR_CNT_2_MASK
02813Ch (4Fh) - MAC_PCU_REG_TSFTHRESH   ;aka MAC_PCU_TSF_THRESHOLD
028140h (50h) - outcommented:MAC_PCU_REG_TSFCAL ;Misc
028144h (51h) - MAC_PCU_REG_PHYERR_EIFS ;aka MAC_PCU_PHY_ERROR_EIFS_MASK
028148h (52h) - outcommented:MAC_PCU_REG_SYNC1 ;Time
02814Ch (53h) - outcommented:MAC_PCU_REG_SYNC2 ;Misc
028150h (54h) - outcommented:MAC_PCU_REG_SYNC3 ;MCAST Addr_L
028154h (55h) - outcommented:MAC_PCU_REG_SYNC4 ;MCAST Addr_U
028158h (56h) - outcommented:MAC_PCU_REG_SYNC5 ;RX Time
02815Ch (57h) - outcommented:MAC_PCU_REG_SYNC6 ;INC
028160h (58h) - outcommented:MAC_PCU_REG_SYNC7 ;Last Time
028164h (59h) - outcommented:MAC_PCU_REG_SYNC8 ;Updated Time
028168h (5Ah) - MAC_PCU_REG_PHYCNT3     ;aka MAC_PCU_PHY_ERR_CNT_3
02816Ch (5Bh) - MAC_PCU_REG_PHYCNTMASK3 ;aka MAC_PCU_PHY_ERR_CNT_3_MASK
028170h (5Ch) - MAC_PCU_REG_BTMODE      ;aka MAC_PCU_BLUETOOTH_MODE
028174h (5Dh) - MAC_PCU_REG_BTWEIGHT    ;aka MAC_PCU_BLUETOOTH_WEIGHTS
028178h (5Eh) - MAC_PCU_REG_HCF         ;aka MAC_PCU_HCF_TIMEOUT
02817Ch (5Fh) - MAC_PCU_REG_BTMODE2     ;aka MAC_PCU_BLUETOOTH_MODE2
028180h (60h..67h) - MAC_PCU_REG_BFCOEF1[0..7]
0281A0h (68h..6Fh) - N/A
0281C0h (70h) - MAC_PCU_REG_BFCOEF2
```

```
0281C4h (71h) - MAC_PCU_REG_KCMASK
0281C8h (72h..73h) - N/A
0281D0h (74h) - MAC_PCU_REG_TXSIFS      ;aka MAC_PCU_TXSIFS
0281D4h (75h..7Ah) - N/A
0281ECh (7Bh) - MAC_PCU_REG_TXOP_X      ;aka MAC_PCU_TXOP_X
0281F0h (7Ch) - MAC_PCU_REG_TXOP_0_3    ;aka MAC_PCU_TXOP_0_3
0281F4h (7Dh) - MAC_PCU_REG_TXOP_4_7    ;aka MAC_PCU_TXOP_4_7
0281F8h (7Eh) - MAC_PCU_REG_TXOP_8_11   ;aka MAC_PCU_TXOP_8_11
0281FCh (7Fh) - MAC_PCU_REG_TXOP_12_15  ;aka MAC_PCU_TXOP_12_15
028200h (80h..87h) - MAC_PCU_REG_GNRCTMR_N[0..7]  ;aka GENERIC_TIMERSxxx?
028220h (88h..8Fh) - MAC_PCU_REG_GNRCTMR_P[0..7]  ;aka GENERIC_TIMERSxxx?
028240h (90h) - MAC_PCU_REG_GNRCTMR_M   ;aka MAC_PCU_GENERIC_TIMERS_MODE
028244h (91h) - MAC_PCU_REG_SLP32_MODE
028248h (92h) - MAC_PCU_REG_SLP32_WAKE
02824Ch (93h) - MAC_PCU_REG_SLP32_TSF_INC
028250h (94h) - MAC_PCU_REG_SLPMIB1
028254h (95h) - MAC_PCU_REG_SLPMIB2
028258h (96h) - MAC_PCU_REG_SLPMIB3
02825Ch (97h) - MAC_PCU_REG_MISCMODE2   ;aka MAC_PCU_MISC_MODE2
028260h (98h) - MAC_PCU_REG_SLP4
028264h (99h) - MAC_PCU_REG_SLP5
028268h (9Ah) - MAC_PCU_REG_MCICTL   ;\
02826Ch (9Bh) - MAC_PCU_REG_MCIISR   ;
028270h (9Ch) - MAC_PCU_REG_MCIIER   ;
028274h (9Dh) - MAC_PCU_REG_MCIWLP   ;
028278h (9Eh) - MAC_PCU_REG_MCIARW   ;
02827Ch (9Fh) - MAC_PCU_REG_MCIARR   ; whatever MCI stuff
028280h (A0h) - MAC_PCU_REG_MCIADW   ;
028284h (A1h) - MAC_PCU_REG_MCIADR   ;
028288h (A2h) - MAC_PCU_REG_MCIFRW   ;
02828Ch (A3h) - MAC_PCU_REG_MCIFRR   ;
028290h (A4h) - MAC_PCU_REG_MCIQRW   ;
028294h (A5h) - MAC_PCU_REG_MCIQRR   ;
028298h (A6h) - MAC_PCU_REG_MCIGRW   ;
02829Ch (A7h) - MAC_PCU_REG_MCIGRR   ;
0282A0h (A8h) - MAC_PCU_REG_MCISTAT  ;/
0282A4h (A9h) - MAC_PCU_REG_BASIC_RATE_SET0 ;aka MAC_PCU_BASIC_RATE_SET0
0282A8h (AAh) - MAC_PCU_REG_BASIC_RATE_SET1 ;aka MAC_PCU_BASIC_RATE_SET1
0282ACh (ABh) - MAC_PCU_REG_BASIC_RATE_SET2 ;aka MAC_PCU_BASIC_RATE_SET2
0282B0h (ACh) - MAC_PCU_REG_SEC_BSSID_L32   ;aka MAC_PCU_BSSID2_L32
0282B4h (ADh) - MAC_PCU_REG_SEC_BSSID_U16   ;aka MAC_PCU_BSSID2_U16
0282B8h (AEh..13Fh) - N/A
028500h (140h..17Fh) - MAC_PCU_REG_FTYPE[0..3Fh]
028600h (180h..19Fh) - N/A
028680h (1A0h..1BFh) - MAC_PCU_REG_ACKSIFSMEM_RESERVED[0..1Fh]
028700h (1C0h..1DFh) - MAC_PCU_REG_DUR[0..1Fh]
028780h (1E0h..1EFh) - N/A
0287C0h (1F0h..1F7h) - MAC_PCU_REG_RTD[0..7]
0287E0h (1F8h..1FFh) - MAC_PCU_REG_DTR[0..7]
028800h (200h..5FFh) - MAC_PCU_REG_KC[0..3FFh]  ;aka KC = KEY_CACHE ?
```

Undocumented hw2 registers

```
029800h ?       Used in hw2, but undocumented... maybe BASEBAND_0 alike hw4?
02A000h ?       Used in hw2, but undocumented... maybe BASEBAND_1 alike hw4?
030000h ?       Used in hw2, but undocumented... maybe mode switch?
```

Note: 030000h seems to be used only in AR6013/AR6014 (not AR6002), maybe RDMA or EFUSE alike hw4, or maybe for atheros/mitsumi mode switch.

DSi Atheros Wifi - Internal I/O Map Summary (hw4)

Overall Summary (hw4.0)

```
004000h 2E8h    (rtc_wlan_reg.h)
008000h 630h    Memory Controller (BCAM) (vmc_wlan_reg.h)
00C000h 14h     (uart_reg.h)
00D000h ..      DBG_UART_BASE_ADDRESS  ;another UART, as above, for debug?
00E000h 38h     (umbox_wlan_reg.h)
010000h 18h     (si_reg.h)
014000h BCh     (gpio_athr_wlan_reg.h)
018000h 12Ch    (mbox_wlan_reg.h)
01A000h 20000h  WLAN_HOST_IF_WINDOW (mbox_wlan_reg.h)
01C000h 748h    (analog_intf_athr_wlan_reg.h)
020000h DCh     WMAC DMA and IRQ   (mac_dma_reg.h)
020800h 244h    WMAC QCU Queue     (mac_dma_reg.h)
021000h 274h    WMAC DCU           (mac_dma_reg.h)
028000h C00h    MAC_PCU            (mac_pcu_reg.h)
029800h 800h    MAC_PCU_BASEBAND_0 (bb_lc_reg.h)
02A000h 1210h   MAC_PCU_BASEBAND_1 (bb_lc_reg.h)
02C000h 1000h   MAC_PCU_BASEBAND_2 (mac_pcu_reg.h)
02D000h 1000h   MAC_PCU_BASEBAND_3 (mac_pcu_reg.h)
02E000h 800h    MAC_PCU_BUF        (mac_pcu_reg.h)
030100h 68h     (rdma_reg.h)
031000h 1000h   (efuse_reg.h)
```

rtc_wlan_reg.h (hw4.0)

```
004000h 4       WLAN_RESET_CONTROL
004004h 4       WLAN_XTAL_CONTROL
004008h 4       WLAN_TCXO_DETECT
00400Ch 4       WLAN_XTAL_TEST
004010h 4       WLAN_QUADRATURE
004014h 4       WLAN_PLL_CONTROL
004018h 4       WLAN_PLL_SETTLE
00401Ch 4       WLAN_XTAL_SETTLE
004020h 4       WLAN_CPU_CLOCK
004024h 4       WLAN_CLOCK_OUT
004028h 4       WLAN_CLOCK_CONTROL
00402Ch 4       WLAN_BIAS_OVERRIDE
004030h 4       WLAN_WDT_CONTROL           ;\
004034h 4       WLAN_WDT_STATUS            ;
004038h 4       WLAN_WDT                   ; Watchdog Timer
00403Ch 4       WLAN_WDT_COUNT             ;
004040h 4       WLAN_WDT_RESET             ;/
004044h 4       WLAN_INT_STATUS            ;-Interrupt Status
004048h 4       WLAN_LF_TIMER0             ;\
00404Ch 4       WLAN_LF_TIMER_COUNT0       ; Low-Freq Timer 0
004050h 4       WLAN_LF_TIMER_CONTROL0     ;
004054h 4       WLAN_LF_TIMER_STATUS0      ;/
004058h 4       WLAN_LF_TIMER1             ;\
00405Ch 4       WLAN_LF_TIMER_COUNT1       ; Low-Freq Timer 1
004060h 4       WLAN_LF_TIMER_CONTROL1     ;
004064h 4       WLAN_LF_TIMER_STATUS1      ;/
004068h 4       WLAN_LF_TIMER2             ;\
00406Ch 4       WLAN_LF_TIMER_COUNT2       ; Low-Freq Timer 2
004070h 4       WLAN_LF_TIMER_CONTROL2     ;
004074h 4       WLAN_LF_TIMER_STATUS2      ;/
004078h 4       WLAN_LF_TIMER3             ;\
00407Ch 4       WLAN_LF_TIMER_COUNT3       ; Low-Freq Timer 3
004080h 4       WLAN_LF_TIMER_CONTROL3     ;
004084h 4       WLAN_LF_TIMER_STATUS3      ;/
004088h 4       WLAN_HF_TIMER              ;\
00408Ch 4       WLAN_HF_TIMER_COUNT        ; High-Freq Timer
004090h 4       WLAN_HF_LF_COUNT      ;<-- ;
004094h 4       WLAN_HF_TIMER_CONTROL      ;
004098h 4       WLAN_HF_TIMER_STATUS       ;/
00409Ch 4       WLAN_RTC_CONTROL           ;\
0040A0h 4       WLAN_RTC_TIME              ;
```

```
0040A4h 4       WLAN_RTC_DATE               ;
0040A8h 4       WLAN_RTC_SET_TIME           ; Real-Time Clock
0040ACh 4       WLAN_RTC_SET_DATE           ;
0040B0h 4       WLAN_RTC_SET_ALARM          ;
0040B4h 4       WLAN_RTC_CONFIG             ;
0040B8h 4       WLAN_RTC_ALARM_STATUS       ;/
0040BCh 4       WLAN_UART_WAKEUP
0040C0h 4       WLAN_RESET_CAUSE
0040C4h 4       WLAN_SYSTEM_SLEEP
0040C8h 4       WLAN_SDIO_WRAPPER
0040CCh 4       WLAN_MAC_SLEEP_CONTROL
0040D0h 4       WLAN_KEEP_AWAKE
0040D4h 4       WLAN_LPO_CAL_TIME                   ;\
0040D8h 4       WLAN_LPO_INIT_DIVIDEND_INT          ;
0040DCh 4       WLAN_LPO_INIT_DIVIDEND_FRACTION     ; LPO
0040E0h 4       WLAN_LPO_CAL                        ;
0040E4h 4       WLAN_LPO_CAL_TEST_CONTROL           ;
0040E8h 4       WLAN_LPO_CAL_TEST_STATUS            ;/
0040ECh 4       WLAN_CHIP_ID                        ;-Chip ID
0040F0h 4       WLAN_DERIVED_RTC_CLK
0040F4h 4       MAC_PCU_SLP32_MODE          ;\
0040F8h 4       MAC_PCU_SLP32_WAKE          ;
0040FCh 4       MAC_PCU_SLP32_INC           ;
004100h 4       MAC_PCU_SLP_MIB1            ;
004104h 4       MAC_PCU_SLP_MIB2            ;
004108h 4       MAC_PCU_SLP_MIB3            ;/
00410Ch 4       WLAN_POWER_REG              ;\located here in hw4.0
004110h 4       WLAN_CORE_CLK_CTRL          ; (other address as in hw2.0)
004114h 4       WLAN_GPIO_WAKEUP_CONTROL    ;/
(below 4118h..42E8h is new in hw4.0, didn't exist in hw2.0)
004118h 4       (WLAN_)HT
00411Ch 4       MAC_PCU_TSF_L32
004120h 4       MAC_PCU_TSF_U32
004124h 4       MAC_PCU_WBTIMER
004128h 1x24    PAD0
004140h 4x16    MAC_PCU_GENERIC_TIMERS[0..15]
004180h 4       MAC_PCU_GENERIC_TIMERS_MODE
004184h 1x60    PAD1
0041C0h 4x16    MAC_PCU_GENERIC_TIMERS2[0..15]
004200h 4       MAC_PCU_GENERIC_TIMERS_MODE2
004204h 4       MAC_PCU_SLP1
004208h 4       MAC_PCU_SLP2
00420Ch 4       MAC_PCU_RESET_TSF
004210h 4       MAC_PCU_TSF_ADD_PLL
004214h 4       SLEEP_RETENTION
004218h 4       BTCOEXCTRL                          ;\
00421Ch 4       WBSYNC_PRIORITY1                    ;
004220h 4       WBSYNC_PRIORITY2                    ;
004224h 4       WBSYNC_PRIORITY3                    ;
004228h 4       BTCOEX0  ;SYNC_DUR                  ;
00422Ch 4       BTCOEX1  ;CLK_THRES                 ;
004230h 4       BTCOEX2  ;FRAME_THRES               ; Bluetooth
004234h 4       BTCOEX3  ;CLK_CNT                   ; Coexistance
004238h 4       BTCOEX4  ;FRAME_CNT                 ;
00423Ch 4       BTCOEX5  ;IDLE_CNT                  ;
004240h 4       BTCOEX6  ;IDLE_RESET_LVL_BITMAP     ;
004244h 4       LOCK                                ;
004248h 4       NOLOCK_PRIORITY                     ;
00424Ch 4       WBSYNC                              ;
004250h 4       WBSYNC1                             ;
004254h 4       WBSYNC2                             ;
004258h 4       WBSYNC3                             ;
00425Ch 4       WB_TIMER_TARGET                     ;
004260h 4       WB_TIMER_SLOP                       ;
004264h 4       BTCOEX_INT_EN                       ;
```

```
  004268h 4       BTCOEX_INT_STAT                     ;
  00426Ch 4       BTPRIORITY_INT_EN                   ;
  004270h 4       BTPRIORITY_INT_STAT                 ;
  004274h 4       BTPRIORITY_STOMP_INT_EN             ;
  004278h 4       BTPRIORITY_STOMP_INT_STAT           ;/
  00427Ch 4       MAC_PCU_BMISS_TIMEOUT
  004280h 4       MAC_PCU_CAB_AWAKE
  004284h 4       LP_PERF_COUNTER
  004288h 4       LP_PERF_LIGHT_SLEEP
  00428Ch 4       LP_PERF_DEEP_SLEEP
  004290h 4       LP_PERF_ON
  004294h 4       ST_64_BIT                           ;\
  004298h 4       MESSAGE_WR                          ; also Bluetooth Coex
  00429Ch 4       MESSAGE_WR_P                        ; related? (sorted as
  0042A0h 4       MESSAGE_RD                          ; so in hw6 files)
  0042A4h 4       MESSAGE_RD_P                        ;/
  0042A8h 4       CHIP_MODE
  0042ACh 4       CLK_REQ_FALL_EDGE
  0042B0h 4       OTP
  0042B4h 4       OTP_STATUS
  0042B8h 4       PMU
  0042BCh 1x4     PAD2
  0042C0h 4x2     PMU_CONFIG[0..1]
  0042C8h 4       PMU_BYPASS
  0042CCh 4       MAC_PCU_TSF2_L32
  0042D0h 4       MAC_PCU_TSF2_U32
  0042D4h 4       MAC_PCU_GENERIC_TIMERS_MODE3
  0042D8h 4       MAC_PCU_DIRECT_CONNECT
  0042DCh 4       THERM_CTRL1
  0042E0h 4       THERM_CTRL2
  0042E4h 4       THERM_CTRL3
  0042E8h -       unused/unspecified
```

vmc_wlan_reg.h (hw4.0)

```
  008000h 4x128   WLAN_MC_BCAM_VALID[0..127]          ;\
  008200h 4x128   WLAN_MC_BCAM_COMPARE[0..127]        ; ROM Patches
  008400h 4x128   WLAN_MC_BCAM_TARGET[0..127]         ;/
  008600h 4       WLAN_APB_ADDR_ERROR_CONTROL         ;\
  008604h 4       WLAN_APB_ADDR_ERROR_STATUS          ; ADDR_ERROR
  008608h 4       WLAN_AHB_ADDR_ERROR_CONTROL         ;
  00860Ch 4       WLAN_AHB_ADDR_ERROR_STATUS          ;/
  008610h 4       WLAN_BCAM_CONFLICT_ERROR
  008614h 4       WLAN_CPU_PERF_CNT
  008618h 4       WLAN_CPU_INST_FETCH
  00861Ch 4       WLAN_CPU_DATA_FETCH
  008620h 4       WLAN_CPU_RAM1_CONFLICT
  008624h 4       WLAN_CPU_RAM2_CONFLICT
  008628h 4       WLAN_CPU_RAM3_CONFLICT
  00862Ch 4       WLAN_CPU_RAM4_CONFLICT
  008630h -       unused/unspecified
```

uart_reg.h (hw4.0)

```
  00C000h 4       UART_DATA
  00C004h 4       UART_CONTROL
  00C008h 4       UART_CLKDIV
  00C00Ch 4       UART_INT
  00C010h 4       UART_INT_EN
  00C014h -       unused/unspecified
  00D000h ..      DBG_UART_BASE_ADDRESS  ;another UART, as above, for debug?
  00Dxxxh -       unused/unspecified
```

umbox_wlan_reg.h (hw4.0)

```
  00E000h 4x2     UMBOX_FIFO[0..1]
  00E008h 4       UMBOX_FIFO_STATUS
```

```
  00E00Ch 4       UMBOX_DMA_POLICY
  00E010h 4       UMBOX0_DMA_RX_DESCRIPTOR_BASE
  00E014h 4       UMBOX0_DMA_RX_CONTROL
  00E018h 4       UMBOX0_DMA_TX_DESCRIPTOR_BASE
  00E01Ch 4       UMBOX0_DMA_TX_CONTROL
  00E020h 4       UMBOX_FIFO_TIMEOUT
  00E024h 4       UMBOX_INT_STATUS
  00E028h 4       UMBOX_INT_ENABLE
  00E02Ch 4       UMBOX_DEBUG
  00E030h 4       UMBOX_FIFO_RESET
  00E034h 4       UMBOX_HCI_FRAMER
  00E038h -       unused/unspecified
```

si_reg.h (hw4.0)

```
  010000h 4       SI_CONFIG
  010004h 4       SI_CS
  010008h 4       SI_TX_DATA0
  01000Ch 4       SI_TX_DATA1
  010010h 4       SI_RX_DATA0
  010014h 4       SI_RX_DATA1
  010018h -       unused/unspecified
```

gpio_athr_wlan_reg.h (hw4.0)

```
  014000h 4       WLAN_GPIO_OUT            ;\GPIO Output Data
  014004h 4       WLAN_GPIO_OUT_W1TS       ; (direct, and Write-1-To-Set/Clr)
  014008h 4       WLAN_GPIO_OUT_W1TC       ;/
  01400Ch 4       WLAN_GPIO_ENABLE         ;\GPIO Output Enable
  014010h 4       WLAN_GPIO_ENABLE_W1TS    ; (direct, and Write-1-To-Set/Clr)
  014014h 4       WLAN_GPIO_ENABLE_W1TC    ;/
  014018h 4       WLAN_GPIO_IN             ;-GPIO Input
  01401Ch 4       WLAN_GPIO_STATUS         ;\GPIO Interrupt Status
  014020h 4       WLAN_GPIO_STATUS_W1TS    ; (direct, and Write-1-To-Set/Clr)
  014024h 4       WLAN_GPIO_STATUS_W1TC    ;/
  014028h 4       WLAN_GPIO_PIN0     ;GPIO0  Bluetooth coex BT_FREQUENCY
  01402Ch 4       WLAN_GPIO_PIN1     ;GPIO1  Bluetooth coex WLAN_ACTIVE
  014030h 4       WLAN_GPIO_PIN2     ;GPIO2  Bluetooth coex BT_ACTIVE
  014034h 4       WLAN_GPIO_PIN3     ;GPIO3  Bluetooth coex BT_PRIORITY
  014038h 4       WLAN_GPIO_PIN4     ;GPIO4  -
  01403Ch 4       WLAN_GPIO_PIN5     ;GPIO5  JTAG TMS input
  014040h 4       WLAN_GPIO_PIN6     ;GPIO6  JTAG TCK input
  014044h 4       WLAN_GPIO_PIN7     ;GPIO7  JTAG TDI input
  014048h 4       WLAN_GPIO_PIN8     ;GPIO8  JTAG TDO output
  01404Ch 4       WLAN_GPIO_PIN9     ;GPIO9  SDIO CMD
  014050h 4       WLAN_GPIO_PIN10    ;GPIO10 SDIO D3
  014054h 4       WLAN_GPIO_PIN11    ;GPIO11 SDIO D2
  014058h 4       WLAN_GPIO_PIN12    ;GPIO12 SDIO D1
  01405Ch 4       WLAN_GPIO_PIN13    ;GPIO13 SDIO D0
  014060h 4       WLAN_GPIO_PIN14    ;GPIO14 SDIO CLK
  014064h 4       WLAN_GPIO_PIN15    ;GPIO15 HCI UART TXD
  014068h 4       WLAN_GPIO_PIN16    ;GPIO16 HCI UART RTS
  01406Ch 4       WLAN_GPIO_PIN17    ;GPIO17 HCI UART RXD
  014070h 4       WLAN_GPIO_PIN18    ;GPIO18 HCI UART CTS
  014074h 4       WLAN_GPIO_PIN19    ;GPIO19 SDIO/GSPI interface select
  014078h 4       WLAN_GPIO_PIN20    ;GPIO20 SDIO/GSPI interface select
  01407Ch 4       WLAN_GPIO_PIN21    ;GPIO21 external input sleep clock
  014080h 4       WLAN_GPIO_PIN22    ;GPIO22 wake on wireless input (WOW)
  014084h 4       WLAN_GPIO_PIN23    ;GPIO23 reference clk output to BT chip
  014088h 4       WLAN_GPIO_PIN24    ;GPIO24 request clk from BT chip
  01408Ch 4       WLAN_GPIO_PIN25    ;GPIO25 request reference clk (CLK_REQ)
  014090h 4       SDIO
  014094h 4       FUNC_BUS
  014098h 4       WL_SOC_APB
  01409Ch 4       WLAN_SIGMA_DELTA
  0140A0h 4       WL_BOOTSTRAP
```

```
0140A4h 4       CLOCK_GPIO
0140A8h 4       WLAN_DEBUG_CONTROL
0140ACh 4       WLAN_DEBUG_INPUT_SEL
0140B0h 4       WLAN_DEBUG_OUT
0140B4h 4       WLAN_RESET_TUPLE_STATUS
0140B8h 4       ANTENNA_SLEEP_CONTROL
0140BCh -       unused/unspecified
```

MBOX Registers (mbox_wlan_reg.h) (hw4.0)

```
018000h 4x4     WLAN_MBOX_FIFO[0..3]
018010h 4       WLAN_MBOX_FIFO_STATUS
018014h 4       WLAN_MBOX_DMA_POLICY
018018h 4       WLAN_MBOX0_DMA_RX_DESCRIPTOR_BASE   ;\
01801Ch 4       WLAN_MBOX0_DMA_RX_CONTROL           ; MBOX 0
018020h 4       WLAN_MBOX0_DMA_TX_DESCRIPTOR_BASE   ;
018024h 4       WLAN_MBOX0_DMA_TX_CONTROL           ;/
018028h 4       WLAN_MBOX1_DMA_RX_DESCRIPTOR_BASE   ;\
01802Ch 4       WLAN_MBOX1_DMA_RX_CONTROL           ; MBOX 1
018030h 4       WLAN_MBOX1_DMA_TX_DESCRIPTOR_BASE   ;
018034h 4       WLAN_MBOX1_DMA_TX_CONTROL           ;/
018038h 4       WLAN_MBOX2_DMA_RX_DESCRIPTOR_BASE   ;\
01803Ch 4       WLAN_MBOX2_DMA_RX_CONTROL           ; MBOX 2
018040h 4       WLAN_MBOX2_DMA_TX_DESCRIPTOR_BASE   ;
018044h 4       WLAN_MBOX2_DMA_TX_CONTROL           ;/
018048h 4       WLAN_MBOX3_DMA_RX_DESCRIPTOR_BASE   ;\
01804Ch 4       WLAN_MBOX3_DMA_RX_CONTROL           ; MBOX 3
018050h 4       WLAN_MBOX3_DMA_TX_DESCRIPTOR_BASE   ;
018054h 4       WLAN_MBOX3_DMA_TX_CONTROL           ;/
018058h 4       WLAN_MBOX_INT_STATUS                ;\Interrupt
01805Ch 4       WLAN_MBOX_INT_ENABLE                ;/
018060h 4       WLAN_INT_HOST            ;IRQ to sdio/host
018064h 1x28    PAD0
018080h 4x8     WLAN_LOCAL_COUNT[0..7]        ;SDIO func1 ?
0180A0h 4x8     WLAN_COUNT_INC[0..7]          ;SDIO func1 ?
0180C0h 4x8     WLAN_LOCAL_SCRATCH[0..7]      ;SDIO func1 ?
0180E0h 4       WLAN_USE_LOCAL_BUS
0180E4h 4       WLAN_SDIO_CONFIG              ;SDIO func0 ?
0180E8h 4       WLAN_MBOX_DEBUG
0180ECh 4       WLAN_MBOX_FIFO_RESET
0180F0h 4x4     WLAN_MBOX_TXFIFO_POP[0..3]
018100h 4x4     WLAN_MBOX_RXFIFO_POP[0..3]
018110h 4       WLAN_SDIO_DEBUG
018114h 4       WLAN_GMBOX0_DMA_RX_DESCRIPTOR_BASE  ;\
018118h 4       WLAN_GMBOX0_DMA_RX_CONTROL          ;
01811Ch 4       WLAN_GMBOX0_DMA_TX_DESCRIPTOR_BASE  ; new (unlike hw2.0)
018120h 4       WLAN_GMBOX0_DMA_TX_CONTROL          ;
018124h 4       WLAN_GMBOX_INT_STATUS               ;
018128h 4       WLAN_GMBOX_INT_ENABLE               ;/
01812Ch 1x7892  PAD1
01A000h 4x2048  WLAN_HOST_IF_WINDOW[0..2047]
```

analog_intf_athr_wlan_reg.h (hw4.0)

```
01C000h 4       RXRF_BIAS1
01C004h 4       RXRF_BIAS2
01C008h 4       RXRF_GAINSTAGES
01C00Ch 4       RXRF_AGC
01C010h 1x48    PAD__0
01C040h 4       TXRF1
01C044h 4       TXRF2
01C048h 4       TXRF3
01C04Ch 4       TXRF4
01C050h 4       TXRF5
01C054h 4       TXRF6
01C058h 4       TXRF7 ;PADRVGNTAB_0..4    ;\
```

```
01C05Ch 4       TXRF8 ;PADRVGNTAB_5..9    ;
01C060h 4       TXRF9 ;PADRVGNTAB_10..14  ;/
01C064h 4       TXRF10
01C068h 4       TXRF11
01C06Ch 4       TXRF12
01C070h 1x16    PAD__1
01C080h 4       SYNTH1
01C084h 4       SYNTH2
01C088h 4       SYNTH3
01C08Ch 4       SYNTH4
01C090h 4       SYNTH5
01C094h 4       SYNTH6
01C098h 4       SYNTH7
01C09Ch 4       SYNTH8
01C0A0h 4       SYNTH9
01C0A4h 4       SYNTH10
01C0A8h 4       SYNTH11
01C0ACh 4       SYNTH12
01C0B0h 4       SYNTH13
01C0B4h 4       SYNTH14
01C0B8h 1x8     PAD__2
01C0C0h 4       BIAS1
01C0C4h 4       BIAS2
01C0C8h 4       BIAS3
01C0CCh 4       BIAS4
01C0D0h 1x48    PAD__3
01C100h 4       RXTX1
01C104h 4       RXTX2
01C108h 4       RXTX3
01C10Ch 1x52    PAD__4
01C140h 4       BB1
01C144h 4       BB2
01C148h 4       BB3
01C14Ch 1x308   PAD__5
01C280h 4       PLLCLKMODA
01C284h 4       PLLCLKMODA2
01C288h 4       TOP
01C28Ch 4       THERM
01C290h 4       XTAL
01C294h 1x236   PAD__6
01C380h 4       RBIST_CNTRL         ;with extra bit in newer revision
01C384h 4       TX_DC_OFFSET
01C388h 4       TX_TONEGEN0
01C38Ch 4       TX_TONEGEN1
01C390h 4       TX_LFTONEGEN0
01C394h 4       TX_LINEAR_RAMP_I
01C398h 4       TX_LINEAR_RAMP_Q
01C39Ch 4       TX_PRBS_MAG
01C3A0h 4       TX_PRBS_SEED_I
01C3A4h 4       TX_PRBS_SEED_Q
01C3A8h 4       CMAC_DC_CANCEL
01C3ACh 4       CMAC_DC_OFFSET
01C3B0h 4       CMAC_CORR
01C3B4h 4       CMAC_POWER
01C3B8h 4       CMAC_CROSS_CORR
01C3BCh 4       CMAC_I2Q2
01C3C0h 4       CMAC_POWER_HPF
01C3C4h 4       RXDAC_SET1
01C3C8h 4       RXDAC_SET2
01C3CCh 4       RXDAC_LONG_SHIFT
01C3D0h 4       CMAC_RESULTS_I
01C3D4h 4       CMAC_RESULTS_Q
01C3D8h 1x872   PAD__7
01C740h 4       PMU1
```

```
  01C744h 4       PMU2
  01C748h -       unused/unspecified
```

mac_dma_reg.h (hw4.0)

```
  020000h 1x8     -
  020008h 4       MAC_DMA_CR      - MAC Control Register
  02000Ch 4       MAC_DMA_RXDP    - MAC receive queue descriptor pointer
  020010h 4       -
  020014h 4       MAC_DMA_CFG     - MAC configuration and status register
  020018h 4       -
  02001Ch 4       -
  020020h 4       MAC_DMA_MIRT    - Maximum rate threshold register
  020024h 4       MAC_DMA_IER     - MAC Interrupt enable register
  020028h 4       MAC_DMA_TIMT    - Transmit Interrupt Mitigation Threshold
  02002Ch 4       MAC_DMA_RIMT    - Receive Interrupt Mitigation Threshold
  020030h 4       MAC_DMA_TXCFG   - MAC tx DMA size config register
  020034h 4       MAC_DMA_RXCFG   - MAC rx DMA size config register
  020038h 4       -
  02003Ch 4       -
  020040h 4       MAC_DMA_MIBC    - MAC MIB control register
  020044h 4       MAC_DMA_TOPS    - MAC timeout prescale count
  020048h 4       MAC_DMA_RXNPTO  - MAC no frame received timeout
  02004Ch 4       MAC_DMA_TXNPTO  - MAC no frame trasmitted timeout
  020050h 4       MAC_DMA_RPGTO   - MAC receive frame gap timeout
  020054h 4       MAC_DMA_RPCNT   - MAC receive frame count limit
  020058h 4       MAC_DMA_MACMISC - MAC miscellaneous control/status register
  02005Ch ..      -
 MAC IRQ...
  020080h 4       MAC_DMA_ISR    - Primary Interrupt Status Register         ;\
  020084h 4       MAC_DMA_ISR_S0 - Secondary Interrupt 0 Status TX OK/DESC   ;
  020088h 4       MAC_DMA_ISR_S1 - Secondary Interrupt 1 Status TX ERR/EOL   ;
  02008Ch 4       MAC_DMA_ISR_S2 - Secondary Interrupt 2 Status TX URN/MISC  ;
  020090h 4       MAC_DMA_ISR_S3 - Secondary Interrupt 3 Status QCBR OVF/URN ;
  020094h 4       MAC_DMA_ISR_S4 - Secondary Interrupt 4 Status QTRIG        ;
  020098h 4       MAC_DMA_ISR_S5 - Secondary Interrupt 5 Status TIMERS       ;/
  02009Ch 4       -
  0200A0h 4       MAC_DMA_IMR    - Primary Interrupt Mask Register           ;\
  0200A4h 4       MAC_DMA_IMR_S0 - Secondary Interrupt 0 Mask TX OK/DESC     ;
  0200A8h 4       MAC_DMA_IMR_S1 - Secondary Interrupt 1 Mask TX ERR/EOL     ;
  0200ACh 4       MAC_DMA_IMR_S2 - Secondary Interrupt 2 Mask TX URN/MISC    ;
  0200B0h 4       MAC_DMA_IMR_S3 - Secondary Interrupt 3 Mask QCBR OVF/URN   ;
  0200B4h 4       MAC_DMA_IMR_S4 - Secondary Interrupt 4 Mask QTRIG          ;
  0200B8h 4       MAC_DMA_IMR_S5 - Secondary Interrupt 5 Mask TIMERS         ;/
  0200BCh 4       -
  0200C0h 4       MAC_DMA_ISR_RAC  - Primary Interrupt Read-and-Clear        ;\
  0200C4h 4       MAC_DMA_ISR_S0_S - Secondary 0 Read-and-Clear TX OK/DESC   ;
  0200C8h 4       MAC_DMA_ISR_S1_S - Secondary 1 Read-and-Clear TX ERR/EOL   ;
  0200CCh 4       MAC_DMA_ISR_S2_S - Secondary 2 Read-and-Clear TX URN/MISC  ;
  0200D0h 4       MAC_DMA_ISR_S3_S - Secondary 3 Read-and-Clear QCBR OVF/URN ;
  0200D4h 4       MAC_DMA_ISR_S4_S - Secondary 4 Read-and-Clear QTRIG        ;
  0200D8h 4       MAC_DMA_ISR_S5_S - Secondary 5 Read-and-Clear TIMERS       ;/
  0200DCh ..      -
 MAC QCU...
  020800h 4x10    MAC_DMA_Q(0..9)_TXDP         ;MAC Transmit Queue descr.ptr
  020828h ..      -
  020840h 4       MAC_DMA_Q_TXE                ;MAC Transmit Queue enable
  020844h ..      -
  020880h 4       MAC_DMA_Q_TXD                ;MAC Transmit Queue disable
  020884h ..      -
  0208C0h 4x10    MAC_DMA_Q(0..9)_CBRCFG       ;MAC CBR configuration
  0208E8h ..      -
  020900h 4x10    MAC_DMA_Q(0..9)_RDYTIMECFG   ;MAC ReadyTime configuration
  020928h ..      -
  020940h 4       MAC_DMA_Q_ONESHOTMAC_DMAM_SC ;MAC OneShotArm set control
```

```
 020944h ..      -
 020980h 4       MAC_DMA_Q_ONESHOTMAC_DMAM_CC ;MAC OneShotArm clear control
 020984h ..      -
 0209C0h 4x10    MAC_DMA_Q(0..9)_MISC         ;MAC Misc QCU settings
 0209E8h ..      -
 020A00h 4x10    MAC_DMA_Q(0..9)_STS          ;MAC Misc QCU status/counter
 020A28h ..      -
 020A40h 4       MAC_DMA_Q_RDYTIMESHDN        ;MAC ReadyTimeShutdown status
 020A44h ..      -
MAC DCU...
 021000h 4x10    MAC_DMA_D(0..9)_QCUMASK - MAC QCU Mask (DCU-to-QCU or so?)
 021028h 8       -
 021030h 4       MAC_DMA_D_GBL_IFS_SIFS  - DCU global SIFS settings
 021034h 12      -
 021040h 4x10    MAC_DMA_D(0..9)_LCL_IFS - MAC DCU-specific IFS settings
 021068h 8       -
 021070h 4       MAC_DMA_D_GBL_IFS_SLOT   - DC global slot interval
 021074h 12      -
 021080h 4x10    MAC_DMA_D(0..9)_RETRY_LIMIT - MAC Retry limits
 0210A8h 8       -
 0210B0h 4       MAC_DMA_D_GBL_IFS_EIFS   - DCU global EIFS setting
 0210B4h 12      -
 0210C0h 4x10    MAC_DMA_D(0..9)_CHNTIME  - MAC ChannelTime settings
 0210E8h 8       -
 0210F0h 4       MAC_DMA_D_GBL_IFS_MISC   - DCU global misc. IFS settings
 0210F4h 12      -
 021100h 4x10    MAC_DMA_D(0..9)_MISC     - MAC Misc DCU-specific settings
 021128h ..      -
 021140h 4       MAC_DMA_D_SEQNUM         - MAC Frame sequence number
 021144h ..      -
 021180h 4x10    MAC_DMA_D(0..9)_EOL     -
 0211A8h ..      -
 021230h 4       MAC_DMA_D_FPCTL         - DCU frame prefetch settings
 021234h ..      -
 021270h 4       MAC_DMA_D_TXPSE         - DCU transmit pause control/status
 021274h ..      -
```

mac_pcu_reg.h (1) (hw4.0)

```
 028000h 4       MAC_PCU_STA_ADDR_L32
 028004h 4       MAC_PCU_STA_ADDR_U16
 028008h 4       MAC_PCU_BSSID_L32
 02800Ch 4       MAC_PCU_BSSID_U16
 028010h 4       MAC_PCU_BCN_RSSI_AVE
 028014h 4       MAC_PCU_ACK_CTS_TIMEOUT
 028018h 4       MAC_PCU_BCN_RSSI_CTL
 02801Ch 4       MAC_PCU_USEC_LATENCY
 028020h 4       PCU_MAX_CFP_DUR
 028024h 4       MAC_PCU_RX_FILTER
 028028h 4       MAC_PCU_MCAST_FILTER_L32
 02802Ch 4       MAC_PCU_MCAST_FILTER_U32
 028030h 4       MAC_PCU_DIAG_SW
 028034h 4       MAC_PCU_TST_ADDAC
 028038h 4       MAC_PCU_DEF_ANTENNA
 02803Ch 4       MAC_PCU_AES_MUTE_MASK_0
 028040h 4       MAC_PCU_AES_MUTE_MASK_1
 028044h 4       MAC_PCU_GATED_CLKS
 028048h 4       MAC_PCU_OBS_BUS_2
 02804Ch 4       MAC_PCU_OBS_BUS_1
 028050h 4       MAC_PCU_DYM_MIMO_PWR_SAVE
 028054h 4       MAC_PCU_LAST_BEACON_TSF
 028058h 4       MAC_PCU_NAV
 02805Ch 4       MAC_PCU_RTS_SUCCESS_CNT
 028060h 4       MAC_PCU_RTS_FAIL_CNT
 028064h 4       MAC_PCU_ACK_FAIL_CNT
```

```
028068h 4       MAC_PCU_FCS_FAIL_CNT
02806Ch 4       MAC_PCU_BEACON_CNT
028070h 4       MAC_PCU_XRMODE
028074h 4       MAC_PCU_XRDEL
028078h 4       MAC_PCU_XRTO
02807Ch 4       MAC_PCU_XRCRP
028080h 4       MAC_PCU_XRSTMP
028084h 4       MAC_PCU_ADDR1_MASK_L32
028088h 4       MAC_PCU_ADDR1_MASK_U16
02808Ch 4       MAC_PCU_TPC
028090h 4       MAC_PCU_TX_FRAME_CNT
028094h 4       MAC_PCU_RX_FRAME_CNT
028098h 4       MAC_PCU_RX_CLEAR_CNT
02809Ch 4       MAC_PCU_CYCLE_CNT
0280A0h 4       MAC_PCU_QUIET_TIME_1
0280A4h 4       MAC_PCU_QUIET_TIME_2
0280A8h 4       MAC_PCU_QOS_NO_ACK
0280ACh 4       MAC_PCU_PHY_ERROR_MASK
0280B0h 4       MAC_PCU_XRLAT
0280B4h 4       MAC_PCU_RXBUF
0280B8h 4       MAC_PCU_MIC_QOS_CONTROL
0280BCh 4       MAC_PCU_MIC_QOS_SELECT
0280C0h 4       MAC_PCU_MISC_MODE
0280C4h 4       MAC_PCU_FILTER_OFDM_CNT
0280C8h 4       MAC_PCU_FILTER_CCK_CNT
0280CCh 4       MAC_PCU_PHY_ERR_CNT_1
0280D0h 4       MAC_PCU_PHY_ERR_CNT_1_MASK
0280D4h 4       MAC_PCU_PHY_ERR_CNT_2
0280D8h 4       MAC_PCU_PHY_ERR_CNT_2_MASK
0280DCh 4       MAC_PCU_TSF_THRESHOLD
0280E0h 4       MAC_PCU_PHY_ERROR_EIFS_MASK
0280E4h 4       MAC_PCU_PHY_ERR_CNT_3
0280E8h 4       MAC_PCU_PHY_ERR_CNT_3_MASK
0280ECh 4       MAC_PCU_BLUETOOTH_MODE
0280F0h 4       MAC_PCU_BLUETOOTH_WEIGHTS
0280F4h 4       MAC_PCU_BLUETOOTH_MODE2
0280F8h 4       MAC_PCU_TXSIFS
0280FCh 4       MAC_PCU_TXOP_X
028100h 4       MAC_PCU_TXOP_0_3
028104h 4       MAC_PCU_TXOP_4_7
028108h 4       MAC_PCU_TXOP_8_11
02810Ch 4       MAC_PCU_TXOP_12_15
028110h 4       MAC_PCU_LOGIC_ANALYZER
028114h 4       MAC_PCU_LOGIC_ANALYZER_32L
028118h 4       MAC_PCU_LOGIC_ANALYZER_16U
02811Ch 4       MAC_PCU_PHY_ERR_CNT_MASK_CONT
028120h 4       MAC_PCU_AZIMUTH_MODE
028124h 4       MAC_PCU_20_40_MODE
028128h 4       MAC_PCU_RX_CLEAR_DIFF_CNT
02812Ch 4       MAC_PCU_SELF_GEN_ANTENNA_MASK
028130h 4       MAC_PCU_BA_BAR_CONTROL
028134h 4       MAC_PCU_LEGACY_PLCP_SPOOF
028138h 4       MAC_PCU_PHY_ERROR_MASK_CONT
02813Ch 4       MAC_PCU_TX_TIMER
028140h 4       MAC_PCU_TXBUF_CTRL
028144h 4       MAC_PCU_MISC_MODE2     ;with extra bit in newer revision
028148h 4       MAC_PCU_ALT_AES_MUTE_MASK
02814Ch 4       MAC_PCU_AZIMUTH_TIME_STAMP
028150h 4       MAC_PCU_MAX_CFP_DUR
028154h 4       MAC_PCU_HCF_TIMEOUT
028158h 4       MAC_PCU_BLUETOOTH_WEIGHTS2
02815Ch 4       MAC_PCU_BLUETOOTH_TSF_BT_ACTIVE
028160h 4       MAC_PCU_BLUETOOTH_TSF_BT_PRIORITY
028164h 4       MAC_PCU_BLUETOOTH_MODE3
028168h 4       MAC_PCU_BLUETOOTH_MODE4
```

```
 02816Ch 1x148   PAD0
 028200h 4x64    MAC_PCU_BT_BT[0..63]
 028300h 4       MAC_PCU_BT_BT_ASYNC
 028304h 4       MAC_PCU_BT_WL_1
 028308h 4       MAC_PCU_BT_WL_2
 02830Ch 4       MAC_PCU_BT_WL_3
 028310h 4       MAC_PCU_BT_WL_4
 028314h 4       MAC_PCU_COEX_EPTA
 028318h 4       MAC_PCU_COEX_LNAMAXGAIN1
 02831Ch 4       MAC_PCU_COEX_LNAMAXGAIN2
 028320h 4       MAC_PCU_COEX_LNAMAXGAIN3
 028324h 4       MAC_PCU_COEX_LNAMAXGAIN4
 028328h 4       MAC_PCU_BASIC_RATE_SET0
 02832Ch 4       MAC_PCU_BASIC_RATE_SET1
 028330h 4       MAC_PCU_BASIC_RATE_SET2
 028334h 4       MAC_PCU_BASIC_RATE_SET3
 028338h 4       MAC_PCU_RX_INT_STATUS0
 02833Ch 4       MAC_PCU_RX_INT_STATUS1
 028340h 4       MAC_PCU_RX_INT_STATUS2
 028344h 4       MAC_PCU_RX_INT_STATUS3
 028348h 4       HT_HALF_GI_RATE1
 02834Ch 4       HT_HALF_GI_RATE2
 028350h 4       HT_FULL_GI_RATE1
 028354h 4       HT_FULL_GI_RATE2
 028358h 4       LEGACY_RATE1
 02835Ch 4       LEGACY_RATE2
 028360h 4       LEGACY_RATE3
 028364h 4       RX_INT_FILTER          ;with extra bit in newer revision
 028368h 4       RX_INT_OVERFLOW
 02836Ch 4       RX_FILTER_THRESH(0)
 028370h 4       RX_FILTER_THRESH1
 028374h 4       RX_PRIORITY_THRESH0
 028378h 4       RX_PRIORITY_THRESH1
 02837Ch 4       RX_PRIORITY_THRESH2
 028380h 4       RX_PRIORITY_THRESH3
 028384h 4       RX_PRIORITY_OFFSET0
 028388h 4       RX_PRIORITY_OFFSET1
 02838Ch 4       RX_PRIORITY_OFFSET2
 028390h 4       RX_PRIORITY_OFFSET3
 028394h 4       RX_PRIORITY_OFFSET4
 028398h 4       RX_PRIORITY_OFFSET5
 02839Ch 4       MAC_PCU_BSSID2_L32
 0283A0h 4       MAC_PCU_BSSID2_U16
 0283A4h 4       MAC_PCU_TSF1_STATUS_L32
 0283A8h 4       MAC_PCU_TSF1_STATUS_U32
 0283ACh 4       MAC_PCU_TSF2_STATUS_L32
 0283B0h 4       MAC_PCU_TSF2_STATUS_U32
 0283B4h 1x76    PAD1
 028400h 4x64    MAC_PCU_TXBUF_BA[0..63]
 028500h 1x768   PAD2
 028800h 4x256   MAC_PCU_KEY_CACHE_1[0..255]
 028C00h 1x3072  PAD3
 029800h 4x512   MAC_PCU_BASEBAND_0[0..511]   ;\aka BB_xxx ports
 02A000h 4x2048  MAC_PCU_BASEBAND_1[0..2047]  ;/(see below)
 02C000h 4x1024  MAC_PCU_BASEBAND_2[0..1023]  ;\
 02D000h 4x1024  MAC_PCU_BASEBAND_3[0..1023]  ; after BB registers
 02E000h 4x512   MAC_PCU_BUF[0..511]          ;/
 02E800h -       unused/unspecified
```

bb_lc_reg.h (hw4.0)

```
 "BASEBAND_0"
 029800h 4       BB_TEST_CONTROLS
 029804h 4       BB_GEN_CONTROLS
 029808h 4       BB_TEST_CONTROLS_STATUS
```

```
02980Ch 4       BB_TIMING_CONTROLS_1
029810h 4       BB_TIMING_CONTROLS_2
029814h 4       BB_TIMING_CONTROLS_3
029818h 4       BB_D2_CHIP_ID
02981Ch 4       BB_ACTIVE
029820h 4       BB_TX_TIMING_1
029824h 4       BB_TX_TIMING_2
029828h 4       BB_TX_TIMING_3
02982Ch 4       BB_ADDAC_PARALLEL_CONTROL
029830h 1x4     PAD__1
029834h 4       BB_XPA_TIMING_CONTROL
029838h 4       BB_MISC_PA_CONTROL
02983Ch 4       BB_TSTDAC_CONSTANT
029840h 4       BB_FIND_SIGNAL_LOW
029844h 4       BB_SETTLING_TIME
029848h 4       BB_GAIN_FORCE_MAX_GAINS_B0
02984Ch 4       BB_GAINS_MIN_OFFSETS_B0
029850h 4       BB_DESIRED_SIGSIZE
029854h 4       BB_TIMING_CONTROL_3A
029858h 4       BB_FIND_SIGNAL
02985Ch 4       BB_AGC
029860h 4       BB_AGC_CONTROL
029864h 4       BB_CCA_B0
029868h 4       BB_SFCORR
02986Ch 4       BB_SELF_CORR_LOW
029870h 1x4     PAD__2
029874h 4       BB_SYNTH_CONTROL
029878h 4       BB_ADDAC_CLK_SELECT
02987Ch 4       BB_PLL_CNTL
029880h 1x128   PAD__3
029900h 4       BB_VIT_SPUR_MASK_A
029904h 4       BB_VIT_SPUR_MASK_B
029908h 4       BB_PILOT_SPUR_MASK
02990Ch 4       BB_CHAN_SPUR_MASK
029910h 4       BB_SPECTRAL_SCAN
029914h 4       BB_ANALOG_POWER_ON_TIME
029918h 4       BB_SEARCH_START_DELAY
02991Ch 4       BB_MAX_RX_LENGTH
029920h 4       BB_TIMING_CONTROL_4
029924h 4       BB_TIMING_CONTROL_5
029928h 4       BB_PHYONLY_WARM_RESET
02992Ch 4       BB_PHYONLY_CONTROL
029930h 1x4     PAD__4
029934h 4       BB_POWERTX_RATE1   ;Power TX 0..3
029938h 4       BB_POWERTX_RATE2   ;Power TX 4..7
02993Ch 4       BB_POWERTX_MAX     ;Power TX Flags
029940h 4       BB_EXTENSION_RADAR
029944h 4       BB_FRAME_CONTROL
029948h 4       BB_TIMING_CONTROL_6
02994Ch 4       BB_SPUR_MASK_CONTROLS
029950h 4       BB_RX_IQ_CORR_B0
029954h 4       BB_RADAR_DETECTION
029958h 4       BB_RADAR_DETECTION_2
02995Ch 4       BB_TX_PHASE_RAMP_B0
029960h 4       BB_SWITCH_TABLE_CHN_B0
029964h 4       BB_SWITCH_TABLE_COM1
029968h 4       BB_CCA_CTRL_2_B0
02996Ch 4       BB_SWITCH_TABLE_COM2
029970h 4       BB_RESTART
029974h 1x4     PAD__5
029978h 4       BB_SCRAMBLER_SEED
02997Ch 4       BB_RFBUS_REQUEST
029980h 1x32    PAD__6
0299A0h 4       BB_TIMING_CONTROL_11
0299A4h 4       BB_MULTICHAIN_ENABLE
```

```
 0299A8h 4       BB_MULTICHAIN_CONTROL
 0299ACh 4       BB_MULTICHAIN_GAIN_CTRL
 0299B0h 1x4     PAD__7
 0299B4h 4       BB_ADC_GAIN_DC_CORR_B0
 0299B8h 4       BB_EXT_CHAN_PWR_THR_1
 0299BCh 4       BB_EXT_CHAN_PWR_THR_2_B0
 0299C0h 4       BB_EXT_CHAN_SCORR_THR
 0299C4h 4       BB_EXT_CHAN_DETECT_WIN
 0299C8h 4       BB_PWR_THR_20_40_DET
 0299CCh 1x4     PAD__8
 0299D0h 4       BB_SHORT_GI_DELTA_SLOPE
 0299D4h 1x8     PAD__9
 0299DCh 4       BB_CHANINFO_CTRL
 0299E0h 4       BB_HEAVY_CLIP_CTRL
 0299E4h 4       BB_HEAVY_CLIP_20
 0299E8h 4       BB_HEAVY_CLIP_40
 0299ECh 4       BB_RIFS_SRCH
 0299F0h 4       BB_IQ_ADC_CAL_MODE
 0299F4h 1x8     PAD__10
 0299FCh 4       BB_PER_CHAIN_CSD
 029A00h 4x128   BB_RX_OCGAIN[0..127]
 029C00h 4       BB_TX_CRC
 029C04h 1x12    PAD__11
 029C10h 4       BB_IQ_ADC_MEAS_0_B0
 029C14h 4       BB_IQ_ADC_MEAS_1_B0
 029C18h 4       BB_IQ_ADC_MEAS_2_B0
 029C1Ch 4       BB_IQ_ADC_MEAS_3_B0
 029C20h 4       BB_RFBUS_GRANT
 029C24h 4       BB_TSTADC
 029C28h 4       BB_TSTDAC
 029C2Ch 1x4     PAD__12
 029C30h 4       BB_ILLEGAL_TX_RATE
 029C34h 4       BB_SPUR_REPORT_B0
 029C38h 4       BB_CHANNEL_STATUS
 029C3Ch 4       BB_RSSI_B0
 029C40h 4       BB_SPUR_EST_CCK_REPORT_B0
 029C44h 1x104   PAD__13  ;(old 1x172)
 029CF0h 4       BB_CHAN_INFO_NOISE_PWR              ;\
 029CF4h 4       BB_CHAN_INFO_GAIN_DIFF              ; located HERE in
 029CF8h 4       BB_CHAN_INFO_FINE_TIMING            ; older revision
 029CFCh 4       BB_CHAN_INFO_GAIN_B0                ; (unlike below)
 029D00h 4x60    BB_CHAN_INFO_CHAN_TAB_B0[0..59]     ;/
 029CACh 4       BB_CHAN_INFO_NOISE_PWR              ;\
 029CB0h 4       BB_CHAN_INFO_GAIN_DIFF              ; located HERE in
 029CB4h 4       BB_CHAN_INFO_FINE_TIMING            ; newer revision
 029CB8h 4       BB_CHAN_INFO_GAIN_B0                ; (unlike above)
 029CBCh 4x60    BB_CHAN_INFO_CHAN_TAB_B0[0..59]     ;/
 029DACh 1x56    PAD__14  ;(old 1x528 at 9DF0h)
 029DE4h 4       BB_PAPRD_AM2AM_MASK                 ;\
 029DE8h 4       BB_PAPRD_AM2PM_MASK                 ;
 029DECh 4       BB_PAPRD_HT40_MASK                  ;
 029DF0h 4       BB_PAPRD_CTRL0                      ; exists ONLY in
 029DF4h 4       BB_PAPRD_CTRL1                      ; newer revision
 029DF8h 4       BB_PA_GAIN123                       ;
 029DFCh 4       BB_PA_GAIN45                        ;
 029E00h 4x8     BB_PAPRD_PRE_POST_SCALE_(0..7)      ;
 029E20h 4x120   BB_PAPRD_MEM_TAB[....]              ;/
"BASEBAND_1"
 02A000h 4       BB_PEAK_DET_CTRL_1
 02A004h 4       BB_PEAK_DET_CTRL_2
 02A008h 4       BB_RX_GAIN_BOUNDS_1
 02A00Ch 4       BB_RX_GAIN_BOUNDS_2
 02A010h 4       BB_PEAK_DET_CAL_CTRL
 02A014h 4       BB_AGC_DIG_DC_CTRL
 02A018h 4       BB_AGC_DIG_DC_STATUS_I_B0
```

```
02A01Ch 4       BB_AGC_DIG_DC_STATUS_Q_B0
02A020h 1x468   PAD__15
02A1F4h 4       BB_BBB_TXFIR_0
02A1F8h 4       BB_BBB_TXFIR_1
02A1FCh 4       BB_BBB_TXFIR_2
02A200h 4       BB_MODES_SELECT
02A204h 4       BB_BBB_TX_CTRL
02A208h 4       BB_BBB_SIG_DETECT
02A20Ch 4       BB_EXT_ATTEN_SWITCH_CTL_B0
02A210h 4       BB_BBB_RX_CTRL_1
02A214h 4       BB_BBB_RX_CTRL_2
02A218h 4       BB_BBB_RX_CTRL_3
02A21Ch 4       BB_BBB_RX_CTRL_4
02A220h 4       BB_BBB_RX_CTRL_5
02A224h 4       BB_BBB_RX_CTRL_6
02A228h 4       BB_BBB_DAGC_CTRL
02A22Ch 4       BB_FORCE_CLKEN_CCK
02A230h 4       BB_RX_CLEAR_DELAY
02A234h 4       BB_POWERTX_RATE3   ;Power TX 1L,2L,2S
02A238h 4       BB_POWERTX_RATE4   ;Power TX 55L,55S,11L,11S
02A23Ch 1x4     PAD__16
02A240h 4       BB_CCK_SPUR_MIT
02A244h 4       BB_PANIC_WATCHDOG_STATUS
02A248h 4       BB_PANIC_WATCHDOG_CTRL_1
02A24Ch 4       BB_PANIC_WATCHDOG_CTRL_2
02A250h 4       BB_IQCORR_CTRL_CCK      ;with extra bit in newer revision
02A254h 4       BB_BLUETOOTH_CNTL
02A258h 4       BB_TPC_1
02A25Ch 4       BB_TPC_2
02A260h 4       BB_TPC_3
02A264h 4       BB_TPC_4_B0
02A268h 4       BB_ANALOG_SWAP
02A26Ch 4       BB_TPC_5_B0
02A270h 4       BB_TPC_6_B0
02A274h 4       BB_TPC_7
02A278h 4       BB_TPC_8
02A27Ch 4       BB_TPC_9
02A280h 4x32    BB_PDADC_TAB_B0[0..31]
02A300h 4x16    BB_CL_TAB_B0[0..15]
02A340h 4       BB_CL_MAP_0_B0
02A344h 4       BB_CL_MAP_1_B0
02A348h 4       BB_CL_MAP_2_B0
02A34Ch 4       BB_CL_MAP_3_B0
02A350h 1x8     PAD__17
02A358h 4       BB_CL_CAL_CTRL
02A35Ch 4       BB_CL_MAP_PAL_0_B0            ;\
02A360h 4       BB_CL_MAP_PAL_1_B0            ; exists ONLY in
02A364h 4       BB_CL_MAP_PAL_2_B0            ; newer revision
02A368h 4       BB_CL_MAP_PAL_3_B0            ;/
02A36Ch 1x28    PAD__18  ;(old 1x44 at A35Ch)
02A388h 4       BB_RIFS
02A38Ch 4       BB_POWERTX_RATE5   ;Power TX HT20_0..3
02A390h 4       BB_POWERTX_RATE6   ;Power TX HT20_4..7
02A394h 4       BB_TPC_10
02A398h 4       BB_TPC_11_B0
02A39Ch 4       BB_CAL_CHAIN_MASK
02A3A0h 1x28    PAD__19
02A3BCh 4       BB_POWERTX_SUB     ;Power TX Sub_for_2chain
02A3C0h 4       BB_POWERTX_RATE7   ;Power TX HT40_0..3
02A3C4h 4       BB_POWERTX_RATE8   ;Power TX HT40_4..7
02A3C8h 4       BB_POWERTX_RATE9   ;Power TX DUP40/EXT20_CCK/OFDM
02A3CCh 4       BB_POWERTX_RATE10  ;Power TX HT20_8..11
02A3D0h 4       BB_POWERTX_RATE11  ;Power TX HT20/40_12/13
02A3D4h 4       BB_POWERTX_RATE12  ;Power TX HT40_8..11
02A3D8h 4       BB_FORCE_ANALOG
```

```
  02A3DCh 4       BB_TPC_12
  02A3E0h 4       BB_TPC_13
  02A3E4h 4       BB_TPC_14
  02A3E8h 4       BB_TPC_15
  02A3ECh 4       BB_TPC_16
  02A3F0h 4       BB_TPC_17
  02A3F4h 4       BB_TPC_18
  02A3F8h 4       BB_TPC_19
  02A3FCh 4       BB_TPC_20
  02A400h 4x32    BB_TX_GAIN_TAB_(1..32)
  02A480h 4x32    BB_TX_GAIN_TAB_PAL_(1..32)
  02A500h 1x24    PAD__20
  02A518h 4x16    BB_CALTX_GAIN_SET_(0,2,4,6,..,28,30)
  02A558h 4x96    BB_TXIQCAL_MEAS_B0[0..95]
  02A6D8h 4       BB_TXIQCAL_START
  02A6DCh 4       BB_TXIQCAL_CONTROL_0
  02A6E0h 4       BB_TXIQCAL_CONTROL_1
  02A6E4h 4       BB_TXIQCAL_CONTROL_2
  02A6E8h 4       BB_TXIQCAL_CONTROL_3
  02A6ECh 4       BB_TXIQ_CORR_COEFF_01_B0
  02A6F0h 4       BB_TXIQ_CORR_COEFF_23_B0
  02A6F4h 4       BB_TXIQ_CORR_COEFF_45_B0
  02A6F8h 4       BB_TXIQ_CORR_COEFF_67_B0
  02A6FCh 4       BB_TXIQ_CORR_COEFF_89_B0
  02A700h 4       BB_TXIQ_CORR_COEFF_AB_B0
  02A704h 4       BB_TXIQ_CORR_COEFF_CD_B0
  02A708h 4       BB_TXIQ_CORR_COEFF_EF_B0
  02A70Ch 4       BB_CAL_RXBB_GAIN_TBL_0
  02A710h 4       BB_CAL_RXBB_GAIN_TBL_4
  02A714h 4       BB_CAL_RXBB_GAIN_TBL_8
  02A718h 4       BB_CAL_RXBB_GAIN_TBL_12
  02A71Ch 4       BB_CAL_RXBB_GAIN_TBL_16
  02A720h 4       BB_CAL_RXBB_GAIN_TBL_20
  02A724h 4       BB_CAL_RXBB_GAIN_TBL_24
  02A728h 4       BB_TXIQCAL_STATUS_B0
  02A72Ch 4       BB_PAPRD_TRAINER_CNTL1        ;\
  02A730h 4       BB_PAPRD_TRAINER_CNTL2        ;
  02A734h 4       BB_PAPRD_TRAINER_CNTL3        ; exists ONLY in
  02A738h 4       BB_PAPRD_TRAINER_CNTL4        ; newer revision
  02A73Ch 4       BB_PAPRD_TRAINER_STAT1        ;
  02A740h 4       BB_PAPRD_TRAINER_STAT2        ;
  02A744h 4       BB_PAPRD_TRAINER_STAT3        ;/
  02A748h 1x144   PAD__21  ;(old 1x172 at A72Ch)
  02A7D8h 4       BB_FCAL_1
  02A7DCh 4       BB_FCAL_2_B0
  02A7E0h 4       BB_RADAR_BW_FILTER
  02A7E4h 4       BB_DFT_TONE_CTRL_B0
  02A7E8h 4       BB_THERM_ADC_1
  02A7ECh 4       BB_THERM_ADC_2
  02A7F0h 4       BB_THERM_ADC_3
  02A7F4h 4       BB_THERM_ADC_4
  02A7F8h 4       BB_TX_FORCED_GAIN
  02A7FCh 4       BB_ECO_CTRL
  02A800h 1x72    PAD__22
  02A848h 4       BB_GAIN_FORCE_MAX_GAINS_B1
  02A84Ch 4       BB_GAINS_MIN_OFFSETS_B1
  02A850h 1x432   PAD__23
  02AA00h 4x128   BB_RX_OCGAIN2[0..127]
  02AC00h 1x1548  PAD__24
  02B20Ch 4       BB_EXT_ATTEN_SWITCH_CTL_B1
  02B210h -       unused/unspecified
```

mac_pcu_reg.h (2) (hw4.0)

```
  02C000h 4x1024  MAC_PCU_BASEBAND_2[0..1023]  ;\
```

```
  02D000h 4x1024  MAC_PCU_BASEBAND_3[0..1023]  ; after BB registers
  02E000h 4x512   MAC_PCU_BUF[0..511]          ;/
  02E800h -       unused/unspecified
```

rdma_reg.h (hw4.0)

```
  030100h 4       DMA_CONFIG
  030104h 4       DMA_CONTROL
  030108h 4       DMA_SRC
  03010Ch 4       DMA_DEST
  030110h 4       DMA_LENGTH
  030114h 4       VMC_BASE
  030118h 4       INDIRECT_REG
  03011Ch 4       INDIRECT_RETURN
  030120h 4x16    RDMA_REGION_(0..15)_
  030160h 4       DMA_STATUS
  030164h 4       DMA_INT_EN
  030168h -       unused/unspecified
```

efuse_reg.h (hw4.0)

```
  031000h 4       EFUSE_WR_ENABLE_REG
  031004h 4       EFUSE_INT_ENABLE_REG
  031008h 4       EFUSE_INT_STATUS_REG
  03100Ch 4       BITMASK_WR_REG
  031010h 4       VDDQ_SETTLE_TIME_REG
  031014h 4       RD_STROBE_PW_REG
  031018h 4       PG_STROBE_PW_REG
  03101Ch 1x2020  PAD0
  031800h 4x512   EFUSE_INTF[0..511]
  032000h -       unused/unspecified
```

DSi Atheros Wifi - Internal I/O Map Summary (hw6)

Overall Summary (hw6.0)

```
  004000h 33Ch    (rtc_soc_reg.h)
  xxx240h 1Ch     (rtc_sync_reg.h)    ;-unknown base address
  005000h 164h    (rtc_wlan_reg.h)
  006000h 264h    (wlan_coex_reg.h)
  007000h 94h     (bt_coex_reg.h)
  008000h ..      MIT (what is that...?) (maybe related to MITSUMI mode?)
  00C000h 14h     (uart_reg.h)
  00D000h ...     DBG_UART (another UART ?)
  00E000h 38h     (umbox_wlan_reg.h)
  010000h 18h     Serial I2C/SPI (si_reg.h)
  010018h 18h     ADDR_ERROR (si_reg.h)
  014000h 170h    (gpio_athr_wlan_reg.h)
  018000h 130h    (mbox_wlan_reg.h)
  01A000h 2000h   WLAN_HOST_IF_WINDOW (mbox_wlan_reg.h)
  01C000h 748h    (analog_intf_athr_wlan_reg.h)
  020000h 130h    (wmac_dma_reg.h)
  020800h 24Ch    (wmac_qcu_reg.h)
  021000h 7FCh    (wmac_dcu_reg.h)
  028000h 1000h   (wmac_pcu_reg.h)
  029800h 3F8h    bb_reg.h (1) - bb_chn_reg_map
  029C00h 24h     bb_reg.h (2) - bb_mrc_reg_map
  029D00h 1Ch     bb_reg.h (3) - bb_bbb_reg_map
  029E00h 400h    bb_reg.h (4) - bb_agc_reg_map
  02A200h 5F8h    bb_reg.h (5) - bb_sm_reg_map
  02A800h 3F8h    bb_reg.h (6) - bb_chn1_reg_map
  02AE00h 400h    bb_reg.h (7) - bb_agc1_reg_map
  02B200h 5F8h    bb_reg.h (8) - bb_sm1_reg_map
  02C800h 400h    bb_reg.h (9) - bb_chn3_reg_map (DUMMY)
```

```
  02CE00h 184h    bb_reg.h (10) - bb_agc3_reg_map (mostly DUMMY)
  02D200h 600h    bb_reg.h (11) - bb_sm3_reg_map (DUMMY)
  02D800h 20h     bb_reg.h (12) - mit_local_reg_map, aka bb_mit_reg_map
  02E000h 4x2048  MAC_PCU_BUF (wmac_pcu_reg.h)
  030000h 1800h   EFUSE (efuse_wlan_reg.h)
  034000h 1Ch     STEREO 0 (stereo_reg.h)
  035000h 58h     (chk_sum_seg_acc_reg.h)
  036000h ?       STEREO 1 (maybe same format as STEREO 0 ?)
  038000h 3Ch     (mmac_reg.h)
  039000h 0Ch     (fpga_reg.h)
  040000h 8       (bridge_intr_reg.h)
  040100h 8       (mii_reg.h)
  040200h 28h     (mdio_reg.h)
  040800h 20h     (bridge_chain_gmac_0_rx_reg.h)
  040C00h 1Ch     (bridge_chain_gmac_0_tx_reg.h)
  050000h ..      SVD (what is that...?)
  054000h ...     (usb_cast_reg.h)    ;<--- located at 54000h (?)
  054100h ..      usb RX chain 0..5 at 00054100h+(0..5)*100h  (?)
  054700h ..      usb TX chain 0..5 at 00054700h+(0..5)*100h  (?)
  054C00h ...     UART2 (yet another UART ?)
  054D00h A8h     (rdma_reg.h)
  054E00h 50h     (athrI2cSlaveApbCsr.h)
  055000h 40h     I2S (mbox_i2s_reg.h)
  056000h ..      I2S_1 (maybe same format as above "mbox_i2s_reg.h"?)
  xxxxxxh A00h    (map_rf_reg.h)      ;\unknown base address
  xxxxxxh 20h     (odin_reg.h)        ;/
```

rtc_soc_reg.h (hw6.0)

```
  004000h 4       SOC_RESET_CONTROL
  004004h 4       SOC_TCXO_DETECT
  004008h 4       SOC_XTAL_TEST
  00400Ch 1x20    PAD0
  004020h 4       SOC_CPU_CLOCK
  004024h 1x4     PAD1
  004028h 4       SOC_CLOCK_CONTROL
  00402Ch 1x4     PAD2
  004030h 4       SOC_WDT_CONTROL             ;\
  004034h 4       SOC_WDT_STATUS              ;
  004038h 4       SOC_WDT                     ; Watchdog Timer
  00403Ch 4       SOC_WDT_COUNT               ;
  004040h 4       SOC_WDT_RESET               ;/
  004044h 4       SOC_INT_STATUS              ;-Interrupt Status
  004048h 4       SOC_LF_TIMER0               ;\
  00404Ch 4       SOC_LF_TIMER_COUNT0         ; Low-Freq Timer
  004050h 4       SOC_LF_TIMER_CONTROL0       ;
  004054h 4       SOC_LF_TIMER_STATUS0        ;/
  004058h 4       SOC_LF_TIMER1               ;\
  00405Ch 4       SOC_LF_TIMER_COUNT1         ; Low-Freq Timer
  004060h 4       SOC_LF_TIMER_CONTROL1       ;
  004064h 4       SOC_LF_TIMER_STATUS1        ;/
  004068h 4       SOC_LF_TIMER2               ;\
  00406Ch 4       SOC_LF_TIMER_COUNT2         ; Low-Freq Timer
  004070h 4       SOC_LF_TIMER_CONTROL2       ;
  004074h 4       SOC_LF_TIMER_STATUS2        ;/
  004078h 4       SOC_LF_TIMER3               ;\
  00407Ch 4       SOC_LF_TIMER_COUNT3         ; Low-Freq Timer
  004080h 4       SOC_LF_TIMER_CONTROL3       ;
  004084h 4       SOC_LF_TIMER_STATUS3        ;/
  004088h 4       SOC_HF_TIMER                ;\
  00408Ch 4       SOC_HF_TIMER_COUNT          ; High-Freq Timer
  004090h 4       SOC_HF_LF_COUNT       ;<--  ;
  004094h 4       SOC_HF_TIMER_CONTROL        ;
  004098h 4       SOC_HF_TIMER_STATUS         ;/
  00409Ch 4       SOC_RTC_CONTROL             ;\
```

```
0040A0h 4       SOC_RTC_TIME                ;
0040A4h 4       SOC_RTC_DATE                ;
0040A8h 4       SOC_RTC_SET_TIME            ; Real-Time Clock
0040ACh 4       SOC_RTC_SET_DATE            ;
0040B0h 4       SOC_RTC_SET_ALARM           ;
0040B4h 4       SOC_RTC_CONFIG              ;
0040B8h 4       SOC_RTC_ALARM_STATUS        ;/
0040BCh 4       SOC_UART_WAKEUP
0040C0h 4       SOC_RESET_CAUSE
0040C4h 4       SOC_SYSTEM_SLEEP
0040C8h 4       SOC_SDIO_WRAPPER
0040CCh 4       SOC_INT_STATUS1
0040D0h 1x4     PAD3
0040D4h 4       SOC_LPO_CAL_TIME                    ;\
0040D8h 4       SOC_LPO_INIT_DIVIDEND_INT           ;
0040DCh 4       SOC_LPO_INIT_DIVIDEND_FRACTION      ; LPO
0040E0h 4       SOC_LPO_CAL                         ;
0040E4h 4       SOC_LPO_CAL_TEST_CONTROL            ;
0040E8h 4       SOC_LPO_CAL_TEST_STATUS             ;/
0040ECh 4       LEGACY_SOC_CHIP_ID                  ;\Chip ID
0040F0h 4       SOC_CHIP_ID                         ;/
0040F4h 1x24    PAD4
00410Ch 4       SOC_POWER_REG
004110h 4       SOC_CORE_CLK_CTRL
004114h 4       SOC_GPIO_WAKEUP_CONTROL
004118h 1x252   PAD5
004214h 4       SLEEP_RETENTION
004218h 1x108   PAD6
004284h 4       LP_PERF_COUNTER                     ;\
004288h 4       LP_PERF_LIGHT_SLEEP                 ; Perf
00428Ch 4       LP_PERF_DEEP_SLEEP                  ;
004290h 4       LP_PERF_ON                          ;/
004294h 1x20    PAD7
0042A8h 4       CHIP_MODE
0042ACh 4       CLK_REQ_FALL_EDGE
0042B0h 4       OTP                                 ;\OTP
0042B4h 4       OTP_STATUS                          ;/
0042B8h 4       PMU
0042BCh 4       PMU_CONFIG
0042C0h 4       PMU_PAREG
0042C4h 4       PMU_BYPASS
0042C8h 1x20    PAD8
0042DCh 4       THERM_CTRL1                         ;\
0042E0h 4       THERM_CTRL2                         ; Therm
0042E4h 4       THERM_CTRL3                         ;/
0042E8h 4       LISTEN_MODE1
0042ECh 4       LISTEN_MODE2
0042F0h 4       AUDIO_PLL_CONFIG
0042F4h 4       AUDIO_PLL_MODULATION
0042F8h 4       AUDIO_PLL_MOD_STEP
0042FCh 4       CURRENT_AUDIO_PLL_MODULATION
004300h 4       ETH_PLL_CONFIG
004304h 4       CPU_PLL_CONFIG
004308h 4       BB_PLL_CONFIG
00430Ch 4       ETH_XMII
004310h 4       USB_PHY_CONFIG
004314h 4       MITSUMI_INT_CONTROL_REG
004318h 4       MITSUMI_INT_STATUS_REG
00431Ch 4       CURRENT_WORKING_MODE
004320h 4       RTC_SLEEP_COUNT
004324h 4       MIT2_VAP
004328h 4       SECOND_HOST_INFT
00432Ch 4       SDIO_HOST
004330h 4       ENTERPRISE_CONFIG
```

```
  004334h 4       RTC_DEBUG_BUS
  004338h 4       RTC_EXT_CLK_BUF
```

rtc_sync_reg.h (hw6.0)

```
  000000h 1x576   PAD__0
  000240h 4       RTC_SYNC_RESET
  000244h 4       RTC_SYNC_STATUS
  000248h 4       RTC_SYNC_DERIVED
  00024Ch 4       RTC_SYNC_FORCE_WAKE
  000250h 4       RTC_SYNC_INTR_CAUSE
  000254h 4       RTC_SYNC_INTR_ENABLE
  000258h 4       RTC_SYNC_INTR_MASK
  00025Ch ..      -
```

rtc_wlan_reg.h (hw6.0)

```
  005000h 4       WLAN_RESET_CONTROL
  005004h 4       WLAN_XTAL_CONTROL
  005008h 4       WLAN_REG_CONTROL0
  00500Ch 4       WLAN_REG_CONTROL1
  005010h 4       WLAN_QUADRATURE
  005014h 4       WLAN_PLL_CONTROL
  005018h 4       WLAN_PLL_SETTLE
  00501Ch 4       WLAN_XTAL_SETTLE
  005020h 4       WLAN_CLOCK_OUT
  005024h 4       WLAN_BIAS_OVERRIDE
  005028h 4       WLAN_RESET_CAUSE
  00502Ch 4       WLAN_SYSTEM_SLEEP
  005030h 4       WLAN_MAC_SLEEP_CONTROL
  005034h 4       WLAN_KEEP_AWAKE
  005038h 4       WLAN_DERIVED_RTC_CLK
  00503Ch 4       MAC_PCU_SLP32_MODE
  005040h 4       MAC_PCU_SLP32_WAKE
  005044h 4       MAC_PCU_SLP32_INC
  005048h 4       MAC_PCU_SLP_MIB1
  00504Ch 4       MAC_PCU_SLP_MIB2
  005050h 4       MAC_PCU_SLP_MIB3
  005054h 4       MAC_PCU_TSF_L32
  005058h 4       MAC_PCU_TSF_U32
  00505Ch 4       MAC_PCU_WBTIMER_0
  005060h 4       MAC_PCU_WBTIMER_1
  005064h 4x16    MAC_PCU_GENERIC_TIMERS[0..15]
  0050A4h 1x24    PAD__0
  0050BCh 4       MAC_PCU_GENERIC_TIMERS_MODE
  0050C0h 4       MAC_PCU_SLP1
  0050C4h 4       MAC_PCU_SLP2
  0050C8h 4       MAC_PCU_SLP3
  0050CCh 4       MAC_PCU_SLP4
  0050D0h 4       MAC_PCU_RESET_TSF
  0050D4h 4       MAC_PCU_TSF2_L32
  0050D8h 4       MAC_PCU_TSF2_U32
  0050DCh 4x16    MAC_PCU_GENERIC_TIMERS2[0..15]
  00511Ch 1x24    PAD__1
  005134h 4       MAC_PCU_GENERIC_TIMERS_MODE2
  005138h 1x12    PAD__2
  005144h 4       MAC_PCU_TSF_THRESHOLD
  005148h 4       WLAN_HT
  00514Ch 1x4     PAD__3
  005150h 4       MAC_PCU_GENERIC_TIMERS_TSF_SEL
  005154h 4       MAC_PCU_BMISS_TIMEOUT
  005158h 4       MAC_PCU_BMISS2_TIMEOUT
  00515Ch 4       RTC_AXI_AHB_BRIDGE
  005160h 4       UNIFIED_MAC_REVID
  005164h ..      -
```

wlan_coex_reg.h (hw6.0)

```
006000h 4       MCI_COMMAND0
006004h 4       MCI_COMMAND1
006008h 4       MCI_COMMAND2
00600Ch 4       MCI_RX_CTRL
006010h 4       MCI_TX_CTRL
006014h 4       MCI_MSG_ATTRIBUTES_TABLE
006018h 4       MCI_SCHD_TABLE_0
00601Ch 4       MCI_SCHD_TABLE_1
006020h 4       MCI_GPM_0
006024h 4       MCI_GPM_1
006028h 4       MCI_INTERRUPT_RAW
00602Ch 4       MCI_INTERRUPT_EN
006030h 4       MCI_REMOTE_CPU_INT
006034h 4       MCI_REMOTE_CPU_INT_EN
006038h 4       MCI_INTERRUPT_RX_MSG_RAW
00603Ch 4       MCI_INTERRUPT_RX_MSG_EN
006040h 4       MCI_CPU_INT
006044h 4       MCI_RX_STATUS
006048h 4       MCI_CONT_STATUS
00604Ch 4       MCI_BT_PRI0
006050h 4       MCI_BT_PRI1
006054h 4       MCI_BT_PRI2
006058h 4       MCI_BT_PRI3
00605Ch 4       MCI_BT_PRI
006060h 4       MCI_WL_FREQ0
006064h 4       MCI_WL_FREQ1
006068h 4       MCI_WL_FREQ2
00606Ch 4       MCI_GAIN
006070h 4       MCI_WBTIMER1
006074h 4       MCI_WBTIMER2
006078h 4       MCI_WBTIMER3
00607Ch 4       MCI_WBTIMER4
006080h 4       MCI_MAXGAIN
006084h 1x40    PAD__0
0060ACh 4       BTCOEX_CTRL
0060B0h 1x156   PAD__1
00614Ch 4       BTCOEX_CTRL2
006150h 1x260   PAD__2
006254h 4       BTCOEX_DBG
006258h 4       MCI_LAST_HW_MSG_HDR
00625Ch 4       MCI_LAST_HW_MSG_BDY
006260h 4       MCI_MAXGAIN_RST
006264h ..      -
```

bt_coex_reg.h (hw6.0)

```
007000h 4       BTCOEXCTRL                      ;\
007004h 4       WBSYNC_PRIORITY1                ;
007008h 4       WBSYNC_PRIORITY2                ;
00700Ch 4       WBSYNC_PRIORITY3                ;
007010h 4       BTCOEX0  ;SYNC_DUR              ;
007014h 4       BTCOEX1  ;CLK_THRES             ;
007018h 4       BTCOEX2  ;FRAME_THRES           ;
00701Ch 4       BTCOEX3  ;CLK_CNT               ; moved from 004218h (hw4)
007020h 4       BTCOEX4  ;FRAME_CNT             ; to 007000h (hw6)
007024h 4       BTCOEX5  ;IDLE_CNT              ;
007028h 4       BTCOEX6  ;IDLE_RESET_LVL_BITMAP ;
00702Ch 4       LOCK                            ;
007030h 4       NOLOCK_PRIORITY                 ;
007034h 4       WBSYNC                          ;
007038h 4       WBSYNC1                         ;
00703Ch 4       WBSYNC2                         ;
007040h 4       WBSYNC3                         ;
```

```
  007044h 4       WB_TIMER_TARGET                 ;
  007048h 4       WB_TIMER_SLOP                   ;
  00704Ch 4       BTCOEX_INT_EN                   ;
  007050h 4       BTCOEX_INT_STAT                 ;
  007054h 4       BTPRIORITY_INT_EN               ;
  007058h 4       BTPRIORITY_INT_STAT             ;
  00705Ch 4       BTPRIORITY_STOMP_INT_EN         ;
  007060h 4       BTPRIORITY_STOMP_INT_STAT       ;/
  007064h 4       ST_64_BIT                       ;\
  007068h 4       MESSAGE_WR                      ; moved from 004294h (hw4)
  00706Ch 4       MESSAGE_WR_P                    ; to 007064h (hw6)
  007070h 4       MESSAGE_RD                      ;
  007074h 4       MESSAGE_RD_P                    ;/
  007078h 4       BTPRIORITY_INT                  ;\
  00707Ch 4       SCO_PARAMS                      ;
  007080h 4       SCO_PRIORITY                    ;
  007084h 4       SCO_SYNC                        ; new, hw6.0 only
  007088h 4       BTCOEX_RAW_STAT                 ;
  00708Ch 4       BTPRIORITY_RAW_STAT             ;
  007090h 4       BTPRIORITY_STOMP_RAW_STAT       ;/
```

uart_reg.h (hw6.0)

```
  00C000h 4       UART_DATA
  00C004h 4       UART_CONTROL
  00C008h 4       UART_CLKDIV
  00C00Ch 4       UART_INT
  00C010h 4       UART_INT_EN
  00C014h ..      -
  00D000h ..      ??
```

umbox_wlan_reg.h (hw6.0)

```
  00E000h 4x2     UMBOX_FIFO[0..1]
  00E008h 4       UMBOX_FIFO_STATUS
  00E00Ch 4       UMBOX_DMA_POLICY
  00E010h 4       UMBOX0_DMA_RX_DESCRIPTOR_BASE
  00E014h 4       UMBOX0_DMA_RX_CONTROL
  00E018h 4       UMBOX0_DMA_TX_DESCRIPTOR_BASE
  00E01Ch 4       UMBOX0_DMA_TX_CONTROL
  00E020h 4       UMBOX_FIFO_TIMEOUT
  00E024h 4       UMBOX_INT_STATUS
  00E028h 4       UMBOX_INT_ENABLE
  00E02Ch 4       UMBOX_DEBUG
  00E030h 4       UMBOX_FIFO_RESET
  00E034h 4       UMBOX_HCI_FRAMER
```

Serial I2C/SPI (si_reg.h) (hw6.0)

```
  010000h 4       SI_CONFIG
  010004h 4       SI_CS
  010008h 4       SI_TX_DATA0
  01000Ch 4       SI_TX_DATA1
  010010h 4       SI_RX_DATA0
  010014h 4       SI_RX_DATA1
```

ADDR_ERROR (si_reg.h) (hw6.0)

```
  010018h 4       WLAN_APB_ADDR_ERROR_CONTROL ;\
  01001Ch 4       WLAN_APB_ADDR_ERROR_STATUS  ; ADDR_ERROR
  010020h 4       WLAN_AHB_ADDR_ERROR_CONTROL ; (located at 8xxxh in hw4)
  010024h 4       WLAN_AHB_ADDR_ERROR_STATUS  ;/
  010028h 4       WLAN_AHB_CONFIG
  01002Ch 4       WLAN_MEMORY_MAP
```

gpio_athr_wlan_reg.h (hw6.0)

```
  014000h 4       WLAN_GPIO_OUT_LOW        ;\
```

```
014004h 4       WLAN_GPIO_OUT_W1TS_LOW   ;
014008h 4       WLAN_GPIO_OUT_W1TC_LOW   ; GPIO Output Data
01400Ch 4       WLAN_GPIO_OUT_HIGH       ; (direct, and Write-1-To-Set/Clr)
014010h 4       WLAN_GPIO_OUT_W1TS_HIGH  ;
014014h 4       WLAN_GPIO_OUT_W1TC_HIGH  ;/
014018h 4       WLAN_GPIO_ENABLE_LOW                ;\
01401Ch 4       WLAN_GPIO_ENABLE_W1TS_LOW           ;
014020h 4       WLAN_GPIO_ENABLE_W1TC_LOW           ; GPIO Output Enable
014024h 4       WLAN_GPIO_ENABLE_HIGH               ; (direct, and Set/Clr)
014028h 4       WLAN_GPIO_ENABLE_W1TS_HIGH          ;
01402Ch 4       WLAN_GPIO_ENABLE_W1TC_HIGH          ;/
014030h 4       WLAN_GPIO_IN_LOW                       ;\
014034h 4       WLAN_GPIO_STATUS_LOW                ;\ ; GPIO Input
014038h 4       WLAN_GPIO_IN_HIGH                   ;  ;/
01403Ch 4       WLAN_GPIO_STATUS_HIGH               ;
014040h 4       WLAN_GPIO_STATUS_W1TS_LOW           ; GPIO Interrupt Status
014044h 4       WLAN_GPIO_STATUS_W1TC_LOW           ; (direct, and Set/Clr)
014048h 4       WLAN_GPIO_STATUS_W1TS_HIGH          ;
01404Ch 4       WLAN_GPIO_STATUS_W1TC_HIGH          ;/
014050h 4       WLAN_GPIO_PIN0   ;GPIO0 or SDIO_CMD
014054h 4       WLAN_GPIO_PIN1   ;GPIO1 or SDIO_D3
014058h 4       WLAN_GPIO_PIN2   ;GPIO2 or SDIO_D2
01405Ch 4       WLAN_GPIO_PIN3   ;GPIO3 or SDIO_D1
014060h 4       WLAN_GPIO_PIN4   ;GPIO4 or SDIO_D0
014064h 4       WLAN_GPIO_PIN5   ;GPIO5 or SDIO_CLK
014068h 4       WLAN_GPIO_PIN6   ;GPIO6
01406Ch 4       WLAN_GPIO_PIN7   ;GPIO7
014070h 4       WLAN_GPIO_PIN8   ;...
014074h 4       WLAN_GPIO_PIN9   ;..
014078h 4       WLAN_GPIO_PIN10
01407Ch 4       WLAN_GPIO_PIN11
014080h 4       WLAN_GPIO_PIN12
014084h 4       WLAN_GPIO_PIN13
014088h 4       WLAN_GPIO_PIN14
01408Ch 4       WLAN_GPIO_PIN15
014090h 4       WLAN_GPIO_PIN16
014094h 4       WLAN_GPIO_PIN17
014098h 4       WLAN_GPIO_PIN18
01409Ch 4       WLAN_GPIO_PIN19
0140A0h 4       WLAN_GPIO_PIN20
0140A4h 4       WLAN_GPIO_PIN21
0140A8h 4       WLAN_GPIO_PIN22
0140ACh 4       WLAN_GPIO_PIN23
0140B0h 4       WLAN_GPIO_PIN24
0140B4h 4       WLAN_GPIO_PIN25
0140B8h 4       WLAN_GPIO_PIN26
0140BCh 4       WLAN_GPIO_PIN27
0140C0h 4       WLAN_GPIO_PIN28
0140C4h 4       WLAN_GPIO_PIN29
0140C8h 4       WLAN_GPIO_PIN30
0140CCh 4       WLAN_GPIO_PIN31
0140D0h 4       WLAN_GPIO_PIN32
0140D4h 4       WLAN_GPIO_PIN33
0140D8h 4       WLAN_GPIO_PIN34
0140DCh 4       WLAN_GPIO_PIN35
0140E0h 4       WLAN_GPIO_PIN36
0140E4h 4       WLAN_GPIO_PIN37
0140E8h 4       WLAN_GPIO_PIN38
0140ECh 4       WLAN_GPIO_PIN39
0140F0h 4       WLAN_GPIO_PIN40
0140F4h 4       WLAN_GPIO_PIN41
0140F8h 4       WLAN_GPIO_PIN42
0140FCh 4       WLAN_GPIO_PIN43
014100h 4       WLAN_GPIO_PIN44
014104h 4       WLAN_GPIO_PIN45
```

```
014108h 4       WLAN_GPIO_PIN46
01410Ch 4       WLAN_GPIO_PIN47
014110h 4       WLAN_GPIO_PIN48
014114h 4       WLAN_GPIO_PIN49
014118h 4       WLAN_GPIO_PIN50
01411Ch 4       WLAN_GPIO_PIN51
014120h 4       WLAN_GPIO_PIN52
014124h 4       WLAN_GPIO_PIN53
014128h 4       WLAN_GPIO_PIN54
01412Ch 4       WLAN_GPIO_PIN55
014130h 4       WLAN_GPIO_PIN56
014134h 4       SDIO
014138h 4       WL_SOC_APB
01413Ch 4       WLAN_SIGMA_DELTA
014140h 4       WL_BOOTSTRAP
014144h 4       CORE_BOOTSTRAP_LOW
014148h 4       CORE_BOOTSTRAP_HIGH
01414Ch 4       WLAN_DEBUG_CONTROL
014150h 4       WLAN_DEBUG_INPUT_SEL
014154h 4       WLAN_DEBUG_OUT
014158h 4       WLAN_RESET_TUPLE_STATUS
01415Ch 4       ANTENNA_CONTROL
014160h 4       SDIO2
014164h 4       SDHC
014168h 4       AMBA_DEBUG_BUS
01416Ch 4       CPU_MBIST
```

MBOX Registers (mbox_wlan_reg.h) (hw6.0)

```
018000h 4x4     WLAN_MBOX_FIFO[0..3]
018010h 4       WLAN_MBOX_FIFO_STATUS
018014h 4       WLAN_MBOX_DMA_POLICY
018018h 4       WLAN_MBOX0_DMA_RX_DESCRIPTOR_BASE   ;\
01801Ch 4       WLAN_MBOX0_DMA_RX_CONTROL           ; MBOX 0
018020h 4       WLAN_MBOX0_DMA_TX_DESCRIPTOR_BASE   ;
018024h 4       WLAN_MBOX0_DMA_TX_CONTROL           ;/
018028h 4       WLAN_MBOX1_DMA_RX_DESCRIPTOR_BASE   ;\
01802Ch 4       WLAN_MBOX1_DMA_RX_CONTROL           ; MBOX 1
018030h 4       WLAN_MBOX1_DMA_TX_DESCRIPTOR_BASE   ;
018034h 4       WLAN_MBOX1_DMA_TX_CONTROL           ;/
018038h 4       WLAN_MBOX2_DMA_RX_DESCRIPTOR_BASE   ;\
01803Ch 4       WLAN_MBOX2_DMA_RX_CONTROL           ; MBOX 2
018040h 4       WLAN_MBOX2_DMA_TX_DESCRIPTOR_BASE   ;
018044h 4       WLAN_MBOX2_DMA_TX_CONTROL           ;/
018048h 4       WLAN_MBOX3_DMA_RX_DESCRIPTOR_BASE   ;\
01804Ch 4       WLAN_MBOX3_DMA_RX_CONTROL           ; MBOX 3
018050h 4       WLAN_MBOX3_DMA_TX_DESCRIPTOR_BASE   ;
018054h 4       WLAN_MBOX3_DMA_TX_CONTROL           ;/
018058h 4       WLAN_MBOX_INT_STATUS                ;\Interrupt
01805Ch 4       WLAN_MBOX_INT_ENABLE                ;/
018060h 4       WLAN_INT_HOST
018064h 1x28    PAD0
018080h 4x8     WLAN_LOCAL_COUNT[0..7]
0180A0h 4x8     WLAN_COUNT_INC[0..7]
0180C0h 4x8     WLAN_LOCAL_SCRATCH[0..7]
0180E0h 4       WLAN_USE_LOCAL_BUS
0180E4h 4       WLAN_SDIO_CONFIG
0180E8h 4       WLAN_MBOX_DEBUG
0180ECh 4       WLAN_MBOX_FIFO_RESET
0180F0h 4x4     WLAN_MBOX_TXFIFO_POP[0..3]
018100h 4x4     WLAN_MBOX_RXFIFO_POP[0..3]
018110h 4       WLAN_SDIO_DEBUG
018114h 4       WLAN_GMBOX0_DMA_RX_DESCRIPTOR_BASE  ;\
018118h 4       WLAN_GMBOX0_DMA_RX_CONTROL          ;
01811Ch 4       WLAN_GMBOX0_DMA_TX_DESCRIPTOR_BASE  ; hw4.0 and hw6.0
```

```
018120h 4       WLAN_GMBOX0_DMA_TX_CONTROL          ;
018124h 4       WLAN_GMBOX_INT_STATUS               ;
018128h 4       WLAN_GMBOX_INT_ENABLE               ;/
01812Ch 4       STE_MODE                            ;<-- hw6.0 only
018130h 1x7888  PAD1
01A000h 4x2048  WLAN_HOST_IF_WINDOW[0..2047]
```

analog_intf_athr_wlan_reg.h (hw6.0)

```
01C000h 4       RXRF_BIAS1
01C004h 4       RXRF_BIAS2
01C008h 4       RXRF_GAINSTAGES
01C00Ch 4       RXRF_AGC
01C010h 1x48    PAD__0
01C040h 4       TXRF1
01C044h 4       TXRF2
01C048h 4       TXRF3
01C04Ch 4       TXRF4
01C050h 4       TXRF5
01C054h 4       TXRF6
01C058h 4       TXRF7
01C05Ch 4       TXRF8
01C060h 4       TXRF9
01C064h 4       TXRF10
01C068h 4       TXRF11
01C06Ch 4       TXRF12
01C070h 1x16    PAD__1
01C080h 4       SYNTH1
01C084h 4       SYNTH2
01C088h 4       SYNTH3
01C08Ch 4       SYNTH4
01C090h 4       SYNTH5
01C094h 4       SYNTH6
01C098h 4       SYNTH7
01C09Ch 4       SYNTH8
01C0A0h 4       SYNTH9
01C0A4h 4       SYNTH10
01C0A8h 4       SYNTH11
01C0ACh 4       SYNTH12
01C0B0h 4       SYNTH13
01C0B4h 4       SYNTH14
01C0B8h 1x8     PAD__2
01C0C0h 4       BIAS1
01C0C4h 4       BIAS2
01C0C8h 4       BIAS3
01C0CCh 4       BIAS4
01C0D0h 1x48    PAD__3
01C100h 4       RXTX1
01C104h 4       RXTX2
01C108h 4       RXTX3
01C10Ch 1x52    PAD__4
01C140h 4       BB1
01C144h 4       BB2
01C148h 4       BB3
01C14Ch 1x308   PAD__5
01C280h 4       PLLCLKMODA
01C284h 4       PLLCLKMODA2
01C288h 4       TOP
01C28Ch 4       THERM
01C290h 4       XTAL
01C294h 1x236   PAD__6
01C380h 4       RBIST_CNTRL
01C384h 4       TX_DC_OFFSET
01C388h 4       TX_TONEGEN0
01C38Ch 4       TX_TONEGEN1
```

```
  01C390h 4       TX_LFTONEGEN0
  01C394h 4       TX_LINEAR_RAMP_I
  01C398h 4       TX_LINEAR_RAMP_Q
  01C39Ch 4       TX_PRBS_MAG
  01C3A0h 4       TX_PRBS_SEED_I
  01C3A4h 4       TX_PRBS_SEED_Q
  01C3A8h 4       CMAC_DC_CANCEL
  01C3ACh 4       CMAC_DC_OFFSET
  01C3B0h 4       CMAC_CORR
  01C3B4h 4       CMAC_POWER
  01C3B8h 4       CMAC_CROSS_CORR
  01C3BCh 4       CMAC_I2Q2
  01C3C0h 4       CMAC_POWER_HPF
  01C3C4h 4       RXDAC_SET1
  01C3C8h 4       RXDAC_SET2
  01C3CCh 4       RXDAC_LONG_SHIFT
  01C3D0h 4       CMAC_RESULTS_I
  01C3D4h 4       CMAC_RESULTS_Q
  01C3D8h 1x872   PAD__7
  01C740h 4       PMU1
  01C744h 4       PMU2
```

wmac_dma_reg.h (hw6.0)

```
  020000h 1x8     PAD__0
  020008h 4       MAC_DMA_CR
  020004h 1x4     PAD__1
  02000Ch 1x4     PAD__1
  020014h 4       MAC_DMA_CFG
  020018h 4       MAC_DMA_RXBUFPTR_THRESH
  02001Ch 4       MAC_DMA_TXDPPTR_THRESH
  020020h 4       MAC_DMA_MIRT
  020024h 4       MAC_DMA_GLOBAL_IER
  020028h 4       MAC_DMA_TIMT_0
  02002Ch 4       MAC_DMA_RIMT
  020030h 4       MAC_DMA_TXCFG
  020034h 4       MAC_DMA_RXCFG
  020038h 4       MAC_DMA_RXJLA
  02003Ch 1x4     PAD__2
  020040h 4       MAC_DMA_MIBC
  020044h 4       MAC_DMA_TOPS
  020048h 4       MAC_DMA_RXNPTO
  02004Ch 4       MAC_DMA_TXNPTO
  020050h 4       MAC_DMA_RPGTO
  020054h 1x4     PAD__3
  020058h 4       MAC_DMA_MACMISC
  02005Ch 4       MAC_DMA_INTER
  020060h 4       MAC_DMA_DATABUF
  020064h 4       MAC_DMA_GTT
  020068h 4       MAC_DMA_GTTM
  02006Ch 4       MAC_DMA_CST
  020070h 4       MAC_DMA_RXDP_SIZE
  020074h 4       MAC_DMA_RX_QUEUE_HP_RXDP
  020078h 4       MAC_DMA_RX_QUEUE_LP_RXDP
  02007Ch 1x4     PAD__4
  020080h 4       MAC_DMA_ISR_P  - Primary Interrupt Status Register         ;\
  020084h 4       MAC_DMA_ISR_S0 - Secondary Interrupt 0 Status TX OK/DESC   ;
  020088h 4       MAC_DMA_ISR_S1 - Secondary Interrupt 1 Status TX ERR/EOL   ;
  02008Ch 4       MAC_DMA_ISR_S2 - Secondary Interrupt 2 Status TX URN/MISC  ;
  020090h 4       MAC_DMA_ISR_S3 - Secondary Interrupt 3 Status QCBR OVF/URN ;
  020094h 4       MAC_DMA_ISR_S4 - Secondary Interrupt 4 Status QTRIG        ;
  020098h 4       MAC_DMA_ISR_S5 - Secondary Interrupt 5 Status TIMERS       ;
  02009Ch 4       MAC_DMA_ISR_S6 - Secondary Interrupt 6 Status UNKNOWN?     ;/
  0200A0h 4       MAC_DMA_IMR_P  - Primary Interrupt Mask Register           ;\
  0200A4h 4       MAC_DMA_IMR_S0 - Secondary Interrupt 0 Mask TX OK/DESC     ;
```

```
  0200A8h 4       MAC_DMA_IMR_S1 - Secondary Interrupt 1 Mask TX ERR/EOL     ;
  0200ACh 4       MAC_DMA_IMR_S2 - Secondary Interrupt 2 Mask TX URN/MISC    ;
  0200B0h 4       MAC_DMA_IMR_S3 - Secondary Interrupt 3 Mask QCBR OVF/URN   ;
  0200B4h 4       MAC_DMA_IMR_S4 - Secondary Interrupt 4 Mask QTRIG          ;
  0200B8h 4       MAC_DMA_IMR_S5 - Secondary Interrupt 5 Mask TIMERS         ;
  0200BCh 4       MAC_DMA_IMR_S6 - Secondary Interrupt 6 Mask UNKNOWN?       ;/
  0200C0h 4       MAC_DMA_ISR_P_RAC - Primary Interrupt Read-and-Clear       ;\
  0200C4h 4       MAC_DMA_ISR_S0_S  - Secondary 0 Read-and-Clr TX OK/DESC    ;
  0200C8h 4       MAC_DMA_ISR_S1_S  - Secondary 1 Read-and-Clr TX ERR/EOL    ;
  0200CCh 4       MAC_DMA_ISR_S6_S  - Secondary 6? Read-and-Clr UNKNOWN? <-- ;
  0200D0h 4       MAC_DMA_ISR_S2_S  - Secondary 2? Read-and-Clr TX URN/MISC  ;
  0200D4h 4       MAC_DMA_ISR_S3_S  - Secondary 3? Read-and-Clr QCBR OVF/URN ;
  0200D8h 4       MAC_DMA_ISR_S4_S  - Secondary 4? Read-and-Clr QTRIG        ;
  0200DCh 4       MAC_DMA_ISR_S5_S  - Secondary 5? Read-and-Clr TIMERS       ;/
  0200E0h 4       MAC_DMA_DMADBG_0
  0200E4h 4       MAC_DMA_DMADBG_1
  0200E8h 4       MAC_DMA_DMADBG_2
  0200ECh 4       MAC_DMA_DMADBG_3
  0200F0h 4       MAC_DMA_DMADBG_4
  0200F4h 4       MAC_DMA_DMADBG_5
  0200F8h 4       MAC_DMA_DMADBG_6
  0200FCh 4       MAC_DMA_DMADBG_7
  020100h 4       MAC_DMA_QCU_TXDP_REMAINING_QCU_7_0
  020104h 4       MAC_DMA_QCU_TXDP_REMAINING_QCU_9_8
  020108h 4       MAC_DMA_TIMT_1       ;note: "MAC_DMA_TIMT_0" is at 020028h
  02010Ch 4       MAC_DMA_TIMT_2
  020110h 4       MAC_DMA_TIMT_3
  020114h 4       MAC_DMA_TIMT_4
  020118h 4       MAC_DMA_TIMT_5
  02011Ch 4       MAC_DMA_TIMT_6
  020120h 4       MAC_DMA_TIMT_7
  020124h 4       MAC_DMA_TIMT_8
  020128h 4       MAC_DMA_TIMT_9
  02012Ch 4       MAC_DMA_CHKACC
```

wmac_qcu_reg.h (hw6.0)

```
  020800h 4x10    MAC_QCU_TXDP[0..9]
  020828h 1x8     PAD__1
  020830h 4       MAC_QCU_STATUS_RING_START
  020834h 4       MAC_QCU_STATUS_RING_END
  020838h 4       MAC_QCU_STATUS_RING_CURRENT
  02083Ch 1x4     PAD__2
  020840h 4       MAC_QCU_TXE
  020844h 1x60    PAD__3
  020880h 4       MAC_QCU_TXD
  020884h 1x60    PAD__4
  0208C0h 4x10    MAC_QCU_CBR[0..9]
  0208E8h 1x24    PAD__5
  020900h 4x10    MAC_QCU_RDYTIME[0..9]
  020928h 1x24    PAD__6
  020940h 4       MAC_QCU_ONESHOT_ARM_SC
  020944h 1x60    PAD__7
  020980h 4       MAC_QCU_ONESHOT_ARM_CC
  020984h 1x60    PAD__8
  0209C0h 4x10    MAC_QCU_MISC[0..9]
  0209E8h 1x24    PAD__9
  020A00h 4x10    MAC_QCU_CNT[0..9]
  020A28h 1x24    PAD__10
  020A40h 4       MAC_QCU_RDYTIME_SHDN
  020A44h 4       MAC_QCU_DESC_CRC_CHK
  020A48h 4       MAC_QCU_EOL
```

wmac_dcu_reg.h (hw6.0)

```
  021000h 4x10    MAC_DCU_QCUMASK[0..9]
```

```
021028h 1x8     PAD__1
021030h 4       MAC_DCU_GBL_IFS_SIFS
021034h 1x4     PAD__2
021038h 4       MAC_DCU_TXFILTER_DCU0_31_0
02103Ch 4       MAC_DCU_TXFILTER_DCU8_31_0
021040h 4x10    MAC_DCU_LCL_IFS[0..9]
021068h 1x8     PAD__3
021070h 4       MAC_DCU_GBL_IFS_SLOT
021074h 1x4     PAD__4
021078h 4       MAC_DCU_TXFILTER_DCU0_63_32
02107Ch 4       MAC_DCU_TXFILTER_DCU8_63_32
021080h 4x10    MAC_DCU_RETRY_LIMIT[0..9]
0210A8h 1x8     PAD__5
0210B0h 4       MAC_DCU_GBL_IFS_EIFS
0210B4h 1x4     PAD__6
0210B8h 4       MAC_DCU_TXFILTER_DCU0_95_64
0210BCh 4       MAC_DCU_TXFILTER_DCU8_95_64
0210C0h 4x10    MAC_DCU_CHANNEL_TIME[0..9]
0210E8h 1x8     PAD__7
0210F0h 4       MAC_DCU_GBL_IFS_MISC
0210F4h 1x4     PAD__8
0210F8h 4       MAC_DCU_TXFILTER_DCU0_127_96
0210FCh 4       MAC_DCU_TXFILTER_DCU8_127_96
021100h 4x10    MAC_DCU_MISC[0..9]
021128h 1x16    PAD__9
021138h 4       MAC_DCU_TXFILTER_DCU1_31_0
02113Ch 4       MAC_DCU_TXFILTER_DCU9_31_0
021140h 4       MAC_DCU_SEQ
021144h 1x52    PAD__10
021178h 4       MAC_DCU_TXFILTER_DCU1_63_32
02117Ch 4       MAC_DCU_TXFILTER_DCU9_63_32
021180h 1x56    PAD__11
0211B8h 4       MAC_DCU_TXFILTER_DCU1_95_64
0211BCh 4       MAC_DCU_TXFILTER_DCU9_95_64
0211C0h 1x56    PAD__12
0211F8h 4       MAC_DCU_TXFILTER_DCU1_127_96
0211FCh 4       MAC_DCU_TXFILTER_DCU9_127_96
021200h 1x56    PAD__13
021238h 4       MAC_DCU_TXFILTER_DCU2_31_0
02123Ch 1x52    PAD__14
021270h 4       MAC_DCU_PAUSE
021274h 1x4     PAD__15
021278h 4       MAC_DCU_TXFILTER_DCU2_63_32
02127Ch 1x52    PAD__16
0212B0h 4       MAC_DCU_WOW_KACFG
0212B4h 1x4     PAD__17
0212B8h 4       MAC_DCU_TXFILTER_DCU2_95_64
0212BCh 1x52    PAD__18
0212F0h 4       MAC_DCU_TXSLOT
0212F4h 1x4     PAD__19
0212F8h 4       MAC_DCU_TXFILTER_DCU2_127_96
0212FCh 1x60    PAD__20
021338h 4       MAC_DCU_TXFILTER_DCU3_31_0          ;\
02133Ch 1x60    PAD__21                             ;
021378h 4       MAC_DCU_TXFILTER_DCU3_63_32         ;
02137Ch 1x60    PAD__22                             ;
0213B8h 4       MAC_DCU_TXFILTER_DCU3_95_64         ;
0213BCh 1x60    PAD__23                             ;
0213F8h 4       MAC_DCU_TXFILTER_DCU3_127_96        ;/
0213FCh 1x60    PAD__24
021438h 4       MAC_DCU_TXFILTER_DCU4_31_0
02143Ch 4       MAC_DCU_TXFILTER_CLEAR
021440h 1x56    PAD__25
021478h 4       MAC_DCU_TXFILTER_DCU4_63_32
02147Ch 4       MAC_DCU_TXFILTER_SET
```

```
021480h 1x56    PAD__26
0214B8h 4       MAC_DCU_TXFILTER_DCU4_95_64
0214BCh 1x60    PAD__27
0214F8h 4       MAC_DCU_TXFILTER_DCU4_127_96
0214FCh 1x60    PAD__28
021538h 4       MAC_DCU_TXFILTER_DCU5_31_0          ;\
02153Ch 1x60    PAD__29                             ;
021578h 4       MAC_DCU_TXFILTER_DCU5_63_32         ;
02157Ch 1x60    PAD__30                             ;
0215B8h 4       MAC_DCU_TXFILTER_DCU5_95_64         ;
0215BCh 1x60    PAD__31                             ;
0215F8h 4       MAC_DCU_TXFILTER_DCU5_127_96        ;/
0215FCh 1x60    PAD__32
021638h 4       MAC_DCU_TXFILTER_DCU6_31_0          ;\
02163Ch 1x60    PAD__33                             ;
021678h 4       MAC_DCU_TXFILTER_DCU6_63_32         ;
02167Ch 1x60    PAD__34                             ;
0216B8h 4       MAC_DCU_TXFILTER_DCU6_95_64         ;
0216BCh 1x60    PAD__35                             ;
0216F8h 4       MAC_DCU_TXFILTER_DCU6_127_96        ;/
0216FCh 1x60    PAD__36
021738h 4       MAC_DCU_TXFILTER_DCU7_31_0          ;\
02173Ch 1x60    PAD__37                             ;
021778h 4       MAC_DCU_TXFILTER_DCU7_63_32         ;
02177Ch 1x60    PAD__38                             ;
0217B8h 4       MAC_DCU_TXFILTER_DCU7_95_64         ;
0217BCh 1x60    PAD__39                             ;
0217F8h 4       MAC_DCU_TXFILTER_DCU7_127_96        ;/
```

wmac_pcu_reg.h (1) (hw6.0)

```
028000h 4       MAC_PCU_STA_ADDR_L32
028004h 4       MAC_PCU_STA_ADDR_U16
028008h 4       MAC_PCU_BSSID_L32
02800Ch 4       MAC_PCU_BSSID_U16
028010h 4       MAC_PCU_BCN_RSSI_AVE
028014h 4       MAC_PCU_ACK_CTS_TIMEOUT
028018h 4       MAC_PCU_BCN_RSSI_CTL
02801Ch 4       MAC_PCU_USEC_LATENCY
028020h 4       MAC_PCU_BCN_RSSI_CTL2
028024h 4       MAC_PCU_BT_WL_1
028028h 4       MAC_PCU_BT_WL_2
02802Ch 4       MAC_PCU_BT_WL_3
028030h 4       MAC_PCU_BT_WL_4
028034h 4       MAC_PCU_COEX_EPTA
028038h 4       MAC_PCU_MAX_CFP_DUR
02803Ch 4       MAC_PCU_RX_FILTER
028040h 4       MAC_PCU_MCAST_FILTER_L32
028044h 4       MAC_PCU_MCAST_FILTER_U32
028048h 4       MAC_PCU_DIAG_SW
02804Ch 1x8     PAD__1
028054h 4       MAC_PCU_TST_ADDAC
028058h 4       MAC_PCU_DEF_ANTENNA
02805Ch 4       MAC_PCU_AES_MUTE_MASK_0
028060h 4       MAC_PCU_AES_MUTE_MASK_1
028064h 4       MAC_PCU_GATED_CLKS
028068h 4       MAC_PCU_OBS_BUS_2
02806Ch 4       MAC_PCU_OBS_BUS_1
028070h 4       MAC_PCU_DYM_MIMO_PWR_SAVE
028074h 1x12    PAD__2
028080h 4       MAC_PCU_LAST_BEACON_TSF
028084h 4       MAC_PCU_NAV
028088h 4       MAC_PCU_RTS_SUCCESS_CNT
02808Ch 4       MAC_PCU_RTS_FAIL_CNT
028090h 4       MAC_PCU_ACK_FAIL_CNT
```

```
028094h 4       MAC_PCU_FCS_FAIL_CNT
028098h 4       MAC_PCU_BEACON_CNT
02809Ch 4       MAC_PCU_BEACON2_CNT
0280A0h 4       MAC_PCU_BASIC_SET
0280A4h 4       MAC_PCU_MGMT_SEQ
0280A8h 4       MAC_PCU_BF_RPT1
0280ACh 4       MAC_PCU_BF_RPT2
0280B0h 4       MAC_PCU_TX_ANT_1
0280B4h 4       MAC_PCU_TX_ANT_2
0280B8h 4       MAC_PCU_TX_ANT_3
0280BCh 4       MAC_PCU_TX_ANT_4
0280C0h 4       MAC_PCU_XRMODE
0280C4h 4       MAC_PCU_XRDEL
0280C8h 4       MAC_PCU_XRTO
0280CCh 4       MAC_PCU_XRCRP
0280D0h 4       MAC_PCU_XRSTMP
0280D4h 1x8     PAD__3
0280DCh 4       MAC_PCU_SELF_GEN_DEFAULT
0280E0h 4       MAC_PCU_ADDR1_MASK_L32
0280E4h 4       MAC_PCU_ADDR1_MASK_U16
0280E8h 4       MAC_PCU_TPC
0280ECh 4       MAC_PCU_TX_FRAME_CNT
0280F0h 4       MAC_PCU_RX_FRAME_CNT
0280F4h 4       MAC_PCU_RX_CLEAR_CNT
0280F8h 4       MAC_PCU_CYCLE_CNT
0280FCh 4       MAC_PCU_QUIET_TIME_1
028100h 4       MAC_PCU_QUIET_TIME_2
028104h 1x4     PAD__4
028108h 4       MAC_PCU_QOS_NO_ACK
02810Ch 4       MAC_PCU_PHY_ERROR_MASK
028110h 4       MAC_PCU_XRLAT
028114h 4       MAC_PCU_RXBUF
028118h 4       MAC_PCU_MIC_QOS_CONTROL
02811Ch 4       MAC_PCU_MIC_QOS_SELECT
028120h 4       MAC_PCU_MISC_MODE
028124h 4       MAC_PCU_FILTER_OFDM_CNT
028128h 4       MAC_PCU_FILTER_CCK_CNT
02812Ch 4       MAC_PCU_PHY_ERR_CNT_1
028130h 4       MAC_PCU_PHY_ERR_CNT_1_MASK
028134h 4       MAC_PCU_PHY_ERR_CNT_2
028138h 4       MAC_PCU_PHY_ERR_CNT_2_MASK
02813Ch 1x8     PAD__5
028144h 4       MAC_PCU_PHY_ERROR_EIFS_MASK
028148h 1x8     PAD__6
028150h 4       MAC_PCU_COEX_LNAMAXGAIN1
028154h 4       MAC_PCU_COEX_LNAMAXGAIN2
028158h 4       MAC_PCU_COEX_LNAMAXGAIN3
02815Ch 4       MAC_PCU_COEX_LNAMAXGAIN4
028160h 1x8     PAD__7
028168h 4       MAC_PCU_PHY_ERR_CNT_3
02816Ch 4       MAC_PCU_PHY_ERR_CNT_3_MASK
028170h 4       MAC_PCU_BLUETOOTH_MODE
028174h 1x4     PAD__8
028178h 4       MAC_PCU_HCF_TIMEOUT
02817Ch 4       MAC_PCU_BLUETOOTH_MODE2
028180h 1x72    PAD__9
0281C8h 4       MAC_PCU_BLUETOOTH_TSF_BT_ACTIVE
0281CCh 4       MAC_PCU_BLUETOOTH_TSF_BT_PRIORITY
0281D0h 4       MAC_PCU_TXSIFS
0281D4h 4       MAC_PCU_BLUETOOTH_MODE3
0281D8h 4       MAC_PCU_BLUETOOTH_MODE4
0281DCh 4       MAC_PCU_BLUETOOTH_MODE5
0281E0h 4       MAC_PCU_BLUETOOTH_WEIGHTS
0281E4h 4       MAC_PCU_BT_BT_ASYNC
0281E8h 1x4     PAD__10
```

```
0281ECh 4       MAC_PCU_TXOP_X
0281F0h 4       MAC_PCU_TXOP_0_3
0281F4h 4       MAC_PCU_TXOP_4_7
0281F8h 4       MAC_PCU_TXOP_8_11
0281FCh 4       MAC_PCU_TXOP_12_15
028200h 4       MAC_PCU_TDMA_TXFRAME_START_TIME_TRIGGER_LSB
028204h 4       MAC_PCU_TDMA_TXFRAME_START_TIME_TRIGGER_MSB
028208h 4       MAC_PCU_TDMA_SLOT_ALERT_CNTL
02820Ch 1x80    PAD__11
02825Ch 4       MAC_PCU_WOW1                ;WOW Misc
028260h 4       MAC_PCU_WOW2                ;WOW AIFS/SLOT/TRY_CNT
028264h 4       MAC_PCU_LOGIC_ANALYZER
028268h 4       MAC_PCU_LOGIC_ANALYZER_32L
02826Ch 4       MAC_PCU_LOGIC_ANALYZER_16U
028270h 4       MAC_PCU_WOW3_BEACON_FAIL    ;WOW Beacon Fail Enable
028274h 4       MAC_PCU_WOW3_BEACON         ;WOW Beacon Timeout
028278h 4       MAC_PCU_WOW3_KEEP_ALIVE     ;WOW Keep-Alive Timeout
02827Ch 4       MAC_PCU_WOW_KA        ;WOW Auto/Fail/BkoffCs Enable/Disable
028280h 1x4     PAD__12
028284h 4       PCU_1US
028288h 4       PCU_KA
02828Ch 4       WOW_EXACT                   ;WOW Exact Length/Offset 1
028290h 1x4     PAD__13
028294h 4       PCU_WOW4                    ;WOW Offset 0..3
028298h 4       PCU_WOW5                    ;WOW Offset 4..7
02829Ch 4       MAC_PCU_PHY_ERR_CNT_MASK_CONT
0282A0h 1x96    PAD__14
028300h 4       MAC_PCU_AZIMUTH_MODE
028304h 1x16    PAD__15
028314h 4       MAC_PCU_AZIMUTH_TIME_STAMP
028318h 4       MAC_PCU_20_40_MODE
02831Ch 4       MAC_PCU_H_XFER_TIMEOUT
028320h 1x8     PAD__16
028328h 4       MAC_PCU_RX_CLEAR_DIFF_CNT
02832Ch 4       MAC_PCU_SELF_GEN_ANTENNA_MASK
028330h 4       MAC_PCU_BA_BAR_CONTROL
028334h 4       MAC_PCU_LEGACY_PLCP_SPOOF
028338h 4       MAC_PCU_PHY_ERROR_MASK_CONT
02833Ch 4       MAC_PCU_TX_TIMER
028340h 4       MAC_PCU_TXBUF_CTRL
028344h 4       MAC_PCU_MISC_MODE2
028348h 4       MAC_PCU_ALT_AES_MUTE_MASK
02834Ch 4       MAC_PCU_WOW6                ;;WOW RX Buf Start Addr (R)
028350h 4       ASYNC_FIFO_REG1
028354h 4       ASYNC_FIFO_REG2
028358h 4       ASYNC_FIFO_REG3
02835Ch 4       MAC_PCU_WOW5                ;WOW RX Abort Enable
028360h 4       MAC_PCU_WOW_LENGTH1         ;WOW Pattern 0..3
028364h 4       MAC_PCU_WOW_LENGTH2         ;WOW Pattern 4..7
028368h 4       WOW_PATTERN_MATCH_LESS_THAN_256_BYTES
02836Ch 1x4     PAD__17
028370h 4       MAC_PCU_WOW4                ;WOW Pattern Enable/Detect
028374h 4       WOW2_EXACT                  ;WOW Exact Length/Offset 2 ;\
028378h 4       PCU_WOW6                    ;WOW Offset 8..11          ;
02837Ch 4       PCU_WOW7                    ;WOW Offset 12..15         ;
028380h 4       MAC_PCU_WOW_LENGTH3         ;WOW Pattern 8..11         ;
028384h 4       MAC_PCU_WOW_LENGTH4         ;WOW Pattern 12..15        ;/
028388h 4       MAC_PCU_LOCATION_MODE_CONTROL
02838Ch 4       MAC_PCU_LOCATION_MODE_TIMER
028390h 1x8     PAD__18
028398h 4       MAC_PCU_BSSID2_L32
02839Ch 4       MAC_PCU_BSSID2_U16
0283A0h 4       MAC_PCU_DIRECT_CONNECT
0283A4h 4       MAC_PCU_TID_TO_AC
0283A8h 4       MAC_PCU_HP_QUEUE
```

```
0283ACh 1x16    PAD__19
0283BCh 4       MAC_PCU_AGC_SATURATION_CNT0
0283C0h 4       MAC_PCU_AGC_SATURATION_CNT1
0283C4h 4       MAC_PCU_AGC_SATURATION_CNT2
0283C8h 4       MAC_PCU_HW_BCN_PROC1
0283CCh 4       MAC_PCU_HW_BCN_PROC2
0283D0h 4       MAC_PCU_MISC_MODE3
0283D4h 4       MAC_PCU_MISC_MODE4
0283D8h 1x4     PAD__20
0283DCh 4       MAC_PCU_PS_FILTER
0283E0h 4       MAC_PCU_BASIC_RATE_SET0
0283E4h 4       MAC_PCU_BASIC_RATE_SET1
0283E8h 4       MAC_PCU_BASIC_RATE_SET2
0283ECh 4       MAC_PCU_BASIC_RATE_SET3
0283F0h 1x16    PAD__21
028400h 4x64    MAC_PCU_TXBUF_BA[0..63]
028500h 4x64    MAC_PCU_BT_BT[0..63]
028600h 4       MAC_PCU_RX_INT_STATUS0
028604h 4       MAC_PCU_RX_INT_STATUS1
028608h 4       MAC_PCU_RX_INT_STATUS2
02860Ch 4       MAC_PCU_RX_INT_STATUS3
028610h 4       HT_HALF_GI_RATE1
028614h 4       HT_HALF_GI_RATE2
028618h 4       HT_FULL_GI_RATE1
02861Ch 4       HT_FULL_GI_RATE2
028620h 4       LEGACY_RATE1
028624h 4       LEGACY_RATE2
028628h 4       LEGACY_RATE3
02862Ch 4       RX_INT_FILTER
028630h 4       RX_INT_OVERFLOW
028634h 4       RX_FILTER_THRESH0
028638h 4       RX_FILTER_THRESH1
02863Ch 4       RX_PRIORITY_THRESH0
028640h 4       RX_PRIORITY_THRESH1
028644h 4       RX_PRIORITY_THRESH2
028648h 4       RX_PRIORITY_THRESH3
02864Ch 4       RX_PRIORITY_OFFSET0
028650h 4       RX_PRIORITY_OFFSET1
028654h 4       RX_PRIORITY_OFFSET2
028658h 4       RX_PRIORITY_OFFSET3
02865Ch 4       RX_PRIORITY_OFFSET4
028660h 4       RX_PRIORITY_OFFSET5
028664h 4       MAC_PCU_LAST_BEACON2_TSF
028668h 4       MAC_PCU_PHY_ERROR_AIFS
02866Ch 4       MAC_PCU_PHY_ERROR_AIFS_MASK
028670h 4       MAC_PCU_FILTER_RSSI_AVE
028674h 4       MAC_PCU_TBD_FILTER
028678h 4       MAC_PCU_BT_ANT_SLEEP_EXTEND
02867Ch 1x388   PAD__22
028800h 4x512   MAC_PCU_KEY_CACHE[0..511]
029000h 1x20480 PAD__23                     ;<-- this includes BB regs
02E000h 4x2048  MAC_PCU_BUF[0..2047]        ;<-- this after BB regs
```

bb_reg.h (1) - bb_chn_reg_map (hw6.0)

```
029800h 4       BB_TIMING_CONTROLS_1
029804h 4       BB_TIMING_CONTROLS_2
029808h 4       BB_TIMING_CONTROLS_3
02980Ch 4       BB_TIMING_CONTROL_4
029810h 4       BB_TIMING_CONTROL_5
029814h 4       BB_TIMING_CONTROL_6
029818h 4       BB_TIMING_CONTROL_11
02981Ch 4       BB_SPUR_MASK_CONTROLS
029820h 4       BB_FIND_SIGNAL_LOW
029824h 4       BB_SFCORR
```

```
029828h 4       BB_SELF_CORR_LOW
02982Ch 4       BB_EXT_CHAN_SCORR_THR
029830h 4       BB_EXT_CHAN_PWR_THR_2_B0
029834h 4       BB_RADAR_DETECTION
029838h 4       BB_RADAR_DETECTION_2
02983Ch 4       BB_EXTENSION_RADAR
029840h 1x64    PAD__0
029880h 4       BB_MULTICHAIN_CONTROL
029884h 4       BB_PER_CHAIN_CSD
029888h 1x24    PAD__1
0298A0h 4       BB_TX_CRC
0298A4h 4       BB_TSTDAC_CONSTANT
0298A8h 4       BB_SPUR_REPORT_B0
0298ACh 1x4     PAD__2
0298B0h 4       BB_TXIQCAL_CONTROL_3
0298B4h 1x8     PAD__3
0298BCh 4       BB_GREEN_TX_CONTROL_1
0298C0h 4       BB_IQ_ADC_MEAS_0_B0
0298C4h 4       BB_IQ_ADC_MEAS_1_B0
0298C8h 4       BB_IQ_ADC_MEAS_2_B0
0298CCh 4       BB_IQ_ADC_MEAS_3_B0
0298D0h 4       BB_TX_PHASE_RAMP_B0
0298D4h 4       BB_ADC_GAIN_DC_CORR_B0
0298D8h 1x4     PAD__4
0298DCh 4       BB_RX_IQ_CORR_B0
0298E0h 1x4     PAD__5
0298E4h 4       BB_PAPRD_AM2AM_MASK
0298E8h 4       BB_PAPRD_AM2PM_MASK
0298ECh 4       BB_PAPRD_HT40_MASK
0298F0h 4       BB_PAPRD_CTRL0_B0
0298F4h 4       BB_PAPRD_CTRL1_B0
0298F8h 4       BB_PA_GAIN123_B0
0298FCh 4       BB_PA_GAIN45_B0
029900h 4       BB_PAPRD_PRE_POST_SCALE_0_B0
029904h 4       BB_PAPRD_PRE_POST_SCALE_1_B0
029908h 4       BB_PAPRD_PRE_POST_SCALE_2_B0
02990Ch 4       BB_PAPRD_PRE_POST_SCALE_3_B0
029910h 4       BB_PAPRD_PRE_POST_SCALE_4_B0
029914h 4       BB_PAPRD_PRE_POST_SCALE_5_B0
029918h 4       BB_PAPRD_PRE_POST_SCALE_6_B0
02991Ch 4       BB_PAPRD_PRE_POST_SCALE_7_B0
029920h 4x120   BB_PAPRD_MEM_TAB_B0[0..119]
029B00h 4x60    BB_CHAN_INFO_CHAN_TAB_B0[0..59]
029BF0h 4       BB_CHN_TABLES_INTF_ADDR
029BF4h 4       BB_CHN_TABLES_INTF_DATA
```

bb_reg.h (2) - bb_mrc_reg_map (hw6.0)

```
029C00h 4       BB_TIMING_CONTROL_3A
029C04h 4       BB_LDPC_CNTL1
029C08h 4       BB_LDPC_CNTL2
029C0Ch 4       BB_PILOT_SPUR_MASK
029C10h 4       BB_CHAN_SPUR_MASK
029C14h 4       BB_SHORT_GI_DELTA_SLOPE
029C18h 4       BB_ML_CNTL1
029C1Ch 4       BB_ML_CNTL2
029C20h 4       BB_TSTADC
```

bb_reg.h (3) - bb_bbb_reg_map (hw6.0)

```
029D00h 4       BB_BBB_RX_CTRL_1
029D04h 4       BB_BBB_RX_CTRL_2
029D08h 4       BB_BBB_RX_CTRL_3
029D0Ch 4       BB_BBB_RX_CTRL_4
029D10h 4       BB_BBB_RX_CTRL_5
```

```
  029D14h 4       BB_BBB_RX_CTRL_6
  029D18h 4       BB_FORCE_CLKEN_CCK
```

bb_reg.h (4) - bb_agc_reg_map (hw6.0)

```
  029E00h 4       BB_SETTLING_TIME
  029E04h 4       BB_GAIN_FORCE_MAX_GAINS_B0
  029E08h 4       BB_GAINS_MIN_OFFSETS
  029E0Ch 4       BB_DESIRED_SIGSIZE
  029E10h 4       BB_FIND_SIGNAL
  029E14h 4       BB_AGC
  029E18h 4       BB_EXT_ATTEN_SWITCH_CTL_B0
  029E1Ch 4       BB_CCA_B0
  029E20h 4       BB_CCA_CTRL_2_B0
  029E24h 4       BB_RESTART
  029E28h 4       BB_MULTICHAIN_GAIN_CTRL
  029E2Ch 4       BB_EXT_CHAN_PWR_THR_1
  029E30h 4       BB_EXT_CHAN_DETECT_WIN
  029E34h 4       BB_PWR_THR_20_40_DET
  029E38h 4       BB_RIFS_SRCH
  029E3Ch 4       BB_PEAK_DET_CTRL_1
  029E40h 4       BB_PEAK_DET_CTRL_2
  029E44h 4       BB_RX_GAIN_BOUNDS_1
  029E48h 4       BB_RX_GAIN_BOUNDS_2
  029E4Ch 4       BB_PEAK_DET_CAL_CTRL
  029E50h 4       BB_AGC_DIG_DC_CTRL
  029E54h 4       BB_BT_COEX_1
  029E58h 4       BB_BT_COEX_2
  029E5Ch 4       BB_BT_COEX_3
  029E60h 4       BB_BT_COEX_4
  029E64h 4       BB_BT_COEX_5
  029E68h 4       BB_REDPWR_CTRL_1
  029E6Ch 4       BB_REDPWR_CTRL_2
  029E70h 1x272   PAD__0
  029F80h 4       BB_RSSI_B0
  029F84h 4       BB_SPUR_EST_CCK_REPORT_B0
  029F88h 4       BB_AGC_DIG_DC_STATUS_I_B0
  029F8Ch 4       BB_AGC_DIG_DC_STATUS_Q_B0
  029F90h 4       BB_DC_CAL_STATUS_B0
  029F94h 1x44    PAD__1
  029FC0h 4       BB_BBB_SIG_DETECT
  029FC4h 4       BB_BBB_DAGC_CTRL
  029FC8h 4       BB_IQCORR_CTRL_CCK
  029FCCh 4       BB_CCK_SPUR_MIT
  029FD0h 4       BB_MRC_CCK_CTRL
  029FD4h 4       BB_CCK_BLOCKER_DET
  029FD8h 1x40    PAD__2
  02A000h 4x128   BB_RX_OCGAIN[0..127]
```

bb_reg.h (5) - bb_sm_reg_map (hw6.0)

```
  02A200h 4       BB_D2_CHIP_ID
  02A204h 4       BB_GEN_CONTROLS
  02A208h 4       BB_MODES_SELECT
  02A20Ch 4       BB_ACTIVE
  02A210h 1x16    PAD__0
  02A220h 4       BB_VIT_SPUR_MASK_A
  02A224h 4       BB_VIT_SPUR_MASK_B
  02A228h 4       BB_SPECTRAL_SCAN
  02A22Ch 4       BB_RADAR_BW_FILTER
  02A230h 4       BB_SEARCH_START_DELAY
  02A234h 4       BB_MAX_RX_LENGTH
  02A238h 4       BB_FRAME_CONTROL
  02A23Ch 4       BB_RFBUS_REQUEST
  02A240h 4       BB_RFBUS_GRANT
  02A244h 4       BB_RIFS
```

```
02A248h 4       BB_SPECTRAL_SCAN_2
02A24Ch 1x4     PAD__1
02A250h 4       BB_RX_CLEAR_DELAY
02A254h 4       BB_ANALOG_POWER_ON_TIME
02A258h 4       BB_TX_TIMING_1
02A25Ch 4       BB_TX_TIMING_2
02A260h 4       BB_TX_TIMING_3
02A264h 4       BB_XPA_TIMING_CONTROL
02A268h 1x24    PAD__2
02A280h 4       BB_MISC_PA_CONTROL
02A284h 4       BB_SWITCH_TABLE_CHN_B0
02A288h 4       BB_SWITCH_TABLE_COM1
02A28Ch 4       BB_SWITCH_TABLE_COM2
02A290h 1x16    PAD__3
02A2A0h 4       BB_MULTICHAIN_ENABLE
02A2A4h 1x28    PAD__4
02A2C0h 4       BB_CAL_CHAIN_MASK
02A2C4h 4       BB_AGC_CONTROL
02A2C8h 4       BB_IQ_ADC_CAL_MODE
02A2CCh 4       BB_FCAL_1
02A2D0h 4       BB_FCAL_2_B0
02A2D4h 4       BB_DFT_TONE_CTRL_B0
02A2D8h 4       BB_CL_CAL_CTRL
02A2DCh 4       BB_CL_MAP_0_B0
02A2E0h 4       BB_CL_MAP_1_B0
02A2E4h 4       BB_CL_MAP_2_B0
02A2E8h 4       BB_CL_MAP_3_B0
02A2ECh 4       BB_CL_MAP_PAL_0_B0
02A2F0h 4       BB_CL_MAP_PAL_1_B0
02A2F4h 4       BB_CL_MAP_PAL_2_B0
02A2F8h 4       BB_CL_MAP_PAL_3_B0
02A2FCh 1x4     PAD__5
02A300h 4x16    BB_CL_TAB_B0[0..15]
02A340h 4       BB_SYNTH_CONTROL
02A344h 4       BB_ADDAC_CLK_SELECT
02A348h 4       BB_PLL_CNTL
02A34Ch 4       BB_ANALOG_SWAP
02A350h 4       BB_ADDAC_PARALLEL_CONTROL
02A354h 1x4     PAD__6
02A358h 4       BB_FORCE_ANALOG
02A35Ch 1x4     PAD__7
02A360h 4       BB_TEST_CONTROLS
02A364h 4       BB_TEST_CONTROLS_STATUS
02A368h 4       BB_TSTDAC
02A36Ch 4       BB_CHANNEL_STATUS
02A370h 4       BB_CHANINFO_CTRL
02A374h 4       BB_CHAN_INFO_NOISE_PWR
02A378h 4       BB_CHAN_INFO_GAIN_DIFF
02A37Ch 4       BB_CHAN_INFO_FINE_TIMING
02A380h 4       BB_CHAN_INFO_GAIN_B0
02A384h 4       BB_SM_HIST
02A388h 1x8     PAD__8
02A390h 4       BB_SCRAMBLER_SEED
02A394h 4       BB_BBB_TX_CTRL
02A398h 4       BB_BBB_TXFIR_0
02A39Ch 4       BB_BBB_TXFIR_1
02A3A0h 4       BB_BBB_TXFIR_2
02A3A4h 4       BB_HEAVY_CLIP_CTRL
02A3A8h 4       BB_HEAVY_CLIP_20
02A3ACh 4       BB_HEAVY_CLIP_40
02A3B0h 4       BB_ILLEGAL_TX_RATE
02A3B4h 1x12    PAD__9
02A3C0h 4       BB_POWERTX_RATE1   ;Power TX 0..3
02A3C4h 4       BB_POWERTX_RATE2   ;Power TX 4..7
02A3C8h 4       BB_POWERTX_RATE3   ;Power TX 1L,2L,2S
```

```
02A3CCh 4       BB_POWERTX_RATE4   ;Power TX 55L,55S,11L,11S
02A3D0h 4       BB_POWERTX_RATE5   ;Power TX HT20_0..3
02A3D4h 4       BB_POWERTX_RATE6   ;Power TX HT20_4..7
02A3D8h 4       BB_POWERTX_RATE7   ;Power TX HT40_0..3
02A3DCh 4       BB_POWERTX_RATE8   ;Power TX HT40_4..7
02A3E0h 4       BB_POWERTX_RATE9   ;Power TX DUP40/EXT20_CCK/OFDM
02A3E4h 4       BB_POWERTX_RATE10  ;Power TX HT20_8..11
02A3E8h 4       BB_POWERTX_RATE11  ;Power TX HT20/40_12/13
02A3ECh 4       BB_POWERTX_RATE12  ;Power TX HT40_8..11
02A3F0h 4       BB_POWERTX_MAX     ;Power TX Flags
02A3F4h 4       BB_POWERTX_SUB     ;Power TX Sub_for_2chain
02A3F8h 4       BB_TPC_1
02A3FCh 4       BB_TPC_2
02A400h 4       BB_TPC_3
02A404h 4       BB_TPC_4_B0
02A408h 4       BB_TPC_5_B0
02A40Ch 4       BB_TPC_6_B0
02A410h 4       BB_TPC_7
02A414h 4       BB_TPC_8
02A418h 4       BB_TPC_9
02A41Ch 4       BB_TPC_10
02A420h 4       BB_TPC_11_B0
02A424h 4       BB_TPC_12
02A428h 4       BB_TPC_13
02A42Ch 4       BB_TPC_14
02A430h 4       BB_TPC_15
02A434h 4       BB_TPC_16
02A438h 4       BB_TPC_17
02A43Ch 4       BB_TPC_18
02A440h 4       BB_TPC_19_B0
02A444h 4       BB_TPC_20
02A448h 4       BB_THERM_ADC_1
02A44Ch 4       BB_THERM_ADC_2
02A450h 4       BB_THERM_ADC_3
02A454h 4       BB_THERM_ADC_4
02A458h 4       BB_TX_FORCED_GAIN
02A45Ch 1x36    PAD__10
02A480h 4x32    BB_PDADC_TAB_B0[0..31]
02A500h 4x32    BB_TX_GAIN_TAB_(1..32)
02A580h 4       BB_RTT_CTRL
02A584h 4       BB_RTT_TABLE_SW_INTF_B0
02A588h 4       BB_RTT_TABLE_SW_INTF_1_B0
02A58Ch 4       BB_TX_GAIN_TAB_1_16_LSB_EXT
02A590h 4       BB_TX_GAIN_TAB_17_32_LSB_EXT
02A594h 1x108   PAD__11
02A600h 4x16    BB_CALTX_GAIN_SET_(0,2,4,6,..,28,30)
02A640h 1x4     PAD__12
02A644h 4       BB_TXIQCAL_CONTROL_0
02A648h 4       BB_TXIQCAL_CONTROL_1
02A64Ch 4       BB_TXIQCAL_CONTROL_2
02A650h 4       BB_TXIQ_CORR_COEFF_01_B0
02A654h 4       BB_TXIQ_CORR_COEFF_23_B0
02A658h 4       BB_TXIQ_CORR_COEFF_45_B0
02A65Ch 4       BB_TXIQ_CORR_COEFF_67_B0
02A660h 4       BB_TXIQ_CORR_COEFF_89_B0
02A664h 4       BB_TXIQ_CORR_COEFF_AB_B0
02A668h 4       BB_TXIQ_CORR_COEFF_CD_B0
02A66Ch 4       BB_TXIQ_CORR_COEFF_EF_B0
02A670h 4       BB_CAL_RXBB_GAIN_TBL_0
02A674h 4       BB_CAL_RXBB_GAIN_TBL_4
02A678h 4       BB_CAL_RXBB_GAIN_TBL_8
02A67Ch 4       BB_CAL_RXBB_GAIN_TBL_12
02A680h 4       BB_CAL_RXBB_GAIN_TBL_16
02A684h 4       BB_CAL_RXBB_GAIN_TBL_20
02A688h 4       BB_CAL_RXBB_GAIN_TBL_24
```

```
02A68Ch 4       BB_TXIQCAL_STATUS_B0
02A690h 4       BB_PAPRD_TRAINER_CNTL1
02A694h 4       BB_PAPRD_TRAINER_CNTL2
02A698h 4       BB_PAPRD_TRAINER_CNTL3
02A69Ch 4       BB_PAPRD_TRAINER_CNTL4
02A6A0h 4       BB_PAPRD_TRAINER_STAT1
02A6A4h 4       BB_PAPRD_TRAINER_STAT2
02A6A8h 4       BB_PAPRD_TRAINER_STAT3
02A6ACh 1x276   PAD__13
02A7C0h 4       BB_WATCHDOG_STATUS
02A7C4h 4       BB_WATCHDOG_CTRL_1
02A7C8h 4       BB_WATCHDOG_CTRL_2
02A7CCh 4       BB_BLUETOOTH_CNTL
02A7D0h 4       BB_PHYONLY_WARM_RESET
02A7D4h 4       BB_PHYONLY_CONTROL
02A7D8h 1x4     PAD__14
02A7DCh 4       BB_ECO_CTRL
02A7E0h 1x16    PAD__15
02A7F0h 4       BB_TABLES_INTF_ADDR_B0
02A7F4h 4       BB_TABLES_INTF_DATA_B0
```

bb_reg.h (6) - bb_chn1_reg_map (hw6.0)

```
02A800h 1x48    PAD__0
02A830h 4       BB_EXT_CHAN_PWR_THR_2_B1
02A834h 1x116   PAD__1
02A8A8h 4       BB_SPUR_REPORT_B1
02A8ACh 1x20    PAD__2
02A8C0h 4       BB_IQ_ADC_MEAS_0_B1
02A8C4h 4       BB_IQ_ADC_MEAS_1_B1
02A8C8h 4       BB_IQ_ADC_MEAS_2_B1
02A8CCh 4       BB_IQ_ADC_MEAS_3_B1
02A8D0h 4       BB_TX_PHASE_RAMP_B1
02A8D4h 4       BB_ADC_GAIN_DC_CORR_B1
02A8D8h 1x4     PAD__3
02A8DCh 4       BB_RX_IQ_CORR_B1
02A8E0h 1x16    PAD__4
02A8F0h 4       BB_PAPRD_CTRL0_B1
02A8F4h 4       BB_PAPRD_CTRL1_B1
02A8F8h 4       BB_PA_GAIN123_B1
02A8FCh 4       BB_PA_GAIN45_B1
02A900h 4       BB_PAPRD_PRE_POST_SCALE_0_B1
02A904h 4       BB_PAPRD_PRE_POST_SCALE_1_B1
02A908h 4       BB_PAPRD_PRE_POST_SCALE_2_B1
02A90Ch 4       BB_PAPRD_PRE_POST_SCALE_3_B1
02A910h 4       BB_PAPRD_PRE_POST_SCALE_4_B1
02A914h 4       BB_PAPRD_PRE_POST_SCALE_5_B1
02A918h 4       BB_PAPRD_PRE_POST_SCALE_6_B1
02A91Ch 4       BB_PAPRD_PRE_POST_SCALE_7_B1
02A920h 4x120   BB_PAPRD_MEM_TAB_B1[0..119]
02AB00h 4x60    BB_CHAN_INFO_CHAN_TAB_B1[0..59]
02ABF0h 4       BB_CHN1_TABLES_INTF_ADDR
02ABF4h 4       BB_CHN1_TABLES_INTF_DATA
```

bb_reg.h (7) - bb_agc1_reg_map (hw6.0)

```
02AE00h 1x4     PAD__0
02AE04h 4       BB_GAIN_FORCE_MAX_GAINS_B1
02AE08h 1x16    PAD__1
02AE18h 4       BB_EXT_ATTEN_SWITCH_CTL_B1
02AE1Ch 4       BB_CCA_B1
02AE20h 4       BB_CCA_CTRL_2_B1
02AE24h 1x348   PAD__2
02AF80h 4       BB_RSSI_B1
02AF84h 4       BB_SPUR_EST_CCK_REPORT_B1
02AF88h 4       BB_AGC_DIG_DC_STATUS_I_B1
```

```
  02AF8Ch 4       BB_AGC_DIG_DC_STATUS_Q_B1
  02AF90h 4       BB_DC_CAL_STATUS_B1
  02AF94h 1x108   PAD__3
  02B000h 4x128   BB_RX_OCGAIN2[0..127]
```

bb_reg.h (8) - bb_sm1_reg_map (hw6.0)

```
  02B200h 1x132   PAD__0
  02B284h 4       BB_SWITCH_TABLE_CHN_B1
  02B288h 1x72    PAD__1
  02B2D0h 4       BB_FCAL_2_B1
  02B2D4h 4       BB_DFT_TONE_CTRL_B1
  02B2D8h 1x4     PAD__2
  02B2DCh 4       BB_CL_MAP_0_B1
  02B2E0h 4       BB_CL_MAP_1_B1
  02B2E4h 4       BB_CL_MAP_2_B1
  02B2E8h 4       BB_CL_MAP_3_B1
  02B2ECh 4       BB_CL_MAP_PAL_0_B1
  02B2F0h 4       BB_CL_MAP_PAL_1_B1
  02B2F4h 4       BB_CL_MAP_PAL_2_B1
  02B2F8h 4       BB_CL_MAP_PAL_3_B1
  02B2FCh 1x4     PAD__3
  02B300h 4x16    BB_CL_TAB_B1[0..15]
  02B340h 1x64    PAD__4
  02B380h 4       BB_CHAN_INFO_GAIN_B1
  02B384h 1x128   PAD__5
  02B404h 4       BB_TPC_4_B1
  02B408h 4       BB_TPC_5_B1
  02B40Ch 4       BB_TPC_6_B1
  02B410h 1x16    PAD__6
  02B420h 4       BB_TPC_11_B1
  02B424h 1x28    PAD__7
  02B440h 4       BB_TPC_19_B1
  02B444h 1x60    PAD__8
  02B480h 4x32    BB_PDADC_TAB_B1[0..31]
  02B500h 1x132   PAD__9
  02B584h 4       BB_RTT_TABLE_SW_INTF_B1
  02B588h 4       BB_RTT_TABLE_SW_INTF_1_B1
  02B58Ch 1x196   PAD__10
  02B650h 4       BB_TXIQ_CORR_COEFF_01_B1
  02B654h 4       BB_TXIQ_CORR_COEFF_23_B1
  02B658h 4       BB_TXIQ_CORR_COEFF_45_B1
  02B65Ch 4       BB_TXIQ_CORR_COEFF_67_B1
  02B660h 4       BB_TXIQ_CORR_COEFF_89_B1
  02B664h 4       BB_TXIQ_CORR_COEFF_AB_B1
  02B668h 4       BB_TXIQ_CORR_COEFF_CD_B1
  02B66Ch 4       BB_TXIQ_CORR_COEFF_EF_B1
  02B670h 1x28    PAD__11
  02B68Ch 4       BB_TXIQCAL_STATUS_B1
  02B690h 1x352   PAD__12
  02B7F0h 4       BB_TABLES_INTF_ADDR_B1
  02B7F4h 4       BB_TABLES_INTF_DATA_B1
```

bb_reg.h (9) - bb_chn3_reg_map (hw6.0)

```
  02C800h 4x256   BB_DUMMY1[0..255]
```

bb_reg.h (10) - bb_agc3_reg_map (hw6.0)

```
  02CE00h 4       BB_DUMMY
  02CE04h 1x380   PAD__0
  02CF80h 4       BB_RSSI_B3
```

bb_reg.h (11) - bb_sm3_reg_map (hw6.0)

```
  02D200h 4x384   BB_DUMMY2[0..383]
```

bb_reg.h (12) - mit_local_reg_map, aka bb_mit_reg_map (hw6.0)

```
02D800h 4       BB_MIT_RF_CNTL
02D804h 4       BB_MIT_CCA_CNTL
02D808h 4       BB_MIT_RSSI_CNTL_1
02D80Ch 4       BB_MIT_RSSI_CNTL_2
02D810h 4       BB_MIT_TX_CNTL
02D814h 4       BB_MIT_RX_CNTL
02D818h 4       BB_MIT_OUT_CNTL
02D81Ch 4       BB_MIT_SPARE_CNTL
```

wmac_pcu_reg.h (2) (hw6.0)

```
02E000h 4x2048  MAC_PCU_BUF[0..2047]
```

efuse_wlan_reg.h (hw6.0)

```
030000h 4       EFUSE_WR_ENABLE_REG
030004h 4       EFUSE_INT_ENABLE_REG
030008h 4       EFUSE_INT_STATUS_REG
03000Ch 4       BITMASK_WR_REG
030010h 4       VDDQ_SETTLE_TIME_REG
030014h 4       VDDQ_HOLD_TIME_REG
030018h 4       RD_STROBE_PW_REG
03001Ch 4       PG_STROBE_PW_REG
030020h 4       PGENB_SETUP_HOLD_TIME_REG
030024h 4       STROBE_PULSE_INTERVAL_REG
030028h 4       CSB_ADDR_LOAD_SETUP_HOLD_REG
03002Ch 1x2004  PAD0
030800h 4x512   EFUSE_INTF0[0..511]
031000h 4x512   EFUSE_INTF1[0..511]
```

stereo_reg.h (hw6.0)

```
034000h 4       STEREO0_CONFIG              ;\Stereo 0
034004h 4       STEREO0_VOLUME              ;/
034008h 4       STEREO_MASTER_CLOCK         ;-Stereo Master
03400Ch 4       STEREO0_TX_SAMPLE_CNT_LSB   ;\
034010h 4       STEREO0_TX_SAMPLE_CNT_MSB   ; Stereo 0
034014h 4       STEREO0_RX_SAMPLE_CNT_LSB   ;
034018h 4       STEREO0_RX_SAMPLE_CNT_MSB   ;/
```

chk_sum_seg_acc_reg.h (hw6.0)

```
035000h 4       CHKSUM_ACC_DMATX_CONTROL0
035004h 4       CHKSUM_ACC_DMATX_CONTROL1
035008h 4       CHKSUM_ACC_DMATX_CONTROL2
03500Ch 4       CHKSUM_ACC_DMATX_CONTROL3
035010h 4       CHKSUM_ACC_DMATX_DESC0
035014h 4       CHKSUM_ACC_DMATX_DESC1
035018h 4       CHKSUM_ACC_DMATX_DESC2
03501Ch 4       CHKSUM_ACC_DMATX_DESC3
035020h 4       CHKSUM_ACC_DMATX_DESC_STATUS
035024h 4       CHKSUM_ACC_DMATX_ARB_CFG
035028h 4       CHKSUM_ACC_RR_PKTCNT01
03502Ch 4       CHKSUM_ACC_RR_PKTCNT23
035030h 4       CHKSUM_ACC_TXST_PKTCNT
035034h 4       CHKSUM_ACC_DMARX_CONTROL
035038h 4       CHKSUM_ACC_DMARX_DESC
03503Ch 4       CHKSUM_ACC_DMARX_DESC_STATUS
035040h 4       CHKSUM_ACC_INTR
035044h 4       CHKSUM_ACC_IMASK
035048h 4       CHKSUM_ACC_ARB_BURST
03504Ch 1x4     PAD__0
035050h 4       CHKSUM_ACC_RESET_DMA
035054h 4       CHKSUM_CONFIG
```

mmac_reg.h (hw6.0)

```
038000h 4       RX_FRAME0
038004h 4       RX_FRAME_0
038008h 4       RX_FRAME1
03800Ch 4       RX_FRAME_1
038010h 4       MMAC_INTERRUPT_RAW
038014h 4       MMAC_INTERRUPT_EN
038018h 4       RX_PARAM1
03801Ch 4       RX_PARAM0
038020h 4       TX_COMMAND0
038024h 4       TX_COMMAND
038028h 4       TX_PARAM
03802Ch 4       BEACON_PARAM
038030h 4       BEACON
038034h 4       TSF_L
038038h 4       TSF_U
```

fpga_reg.h (hw6.0)

```
039000h 4       FPGA_REG1
039004h 4       FPGA_REG2
039008h 4       FPGA_REG4
```

bridge_intr_reg.h (hw6.0)

```
040000h 4       INTERRUPT
040004h 4       INTERRUPT_MASK
```

mii_reg.h (hw6.0)

```
040100h 4       MII0_CNTL
040104h 4       STAT_CNTL
```

mdio_reg.h (hw6.0)

```
040200h 4x8     MDIO_REG[0..7]
040220h 4       MDIO_ISR
040224h 4       PHY_ADDR
```

bridge_chain_gmac_0_rx_reg.h (hw6.0)

```
040800h 4       GMAC_RX_0_DESC_START_ADDRESS
040804h 4       GMAC_RX_0_DMA_START
040808h 4       GMAC_RX_0_BURST_SIZE
04080Ch 4       GMAC_RX_0_PKT_OFFSET
040810h 4       GMAC_RX_0_CHECKSUM
040814h 4       GMAC_RX_0_DBG_RX
040818h 4       GMAC_RX_0_DBG_RX_CUR_ADDR
04081Ch 4       GMAC_RX_0_DATA_SWAP
```

bridge_chain_gmac_0_tx_reg.h (hw6.0)

```
040C00h 4       GMAC_TX_0_DESC_START_ADDRESS
040C04h 4       GMAC_TX_0_DMA_START
040C08h 4       GMAC_TX_0_INTERRUPT_LIMIT
040C0Ch 4       GMAC_TX_0_BURST_SIZE
040C10h 4       GMAC_TX_0_DBG_TX
040C14h 4       GMAC_TX_0_DBG_TX_CUR_ADDR
040C18h 4       GMAC_TX_0_DATA_SWAP
```

usb_cast_reg.h (hw6.0)

```
054000h 4       ENDP0
054004h 1x4     PAD0
054008h 4       OUT1ENDP
05400Ch 4       IN1ENDP
054010h 4       OUT2ENDP
054014h 4       IN2ENDP
```

```
  054018h 4       OUT3ENDP
  05401Ch 4       IN3ENDP
  054020h 4       OUT4ENDP
  054024h 4       IN4ENDP
  054028h 4       OUT5ENDP
  05402Ch 4       IN5ENDP
  054030h 1x92    PAD1
  05408Ch 4       USBMODESTATUS
  054090h 1x248   PAD2
  054188h 4       EPIRQ
  05418Ch 4       USBIRQ
  054190h 1x4     PAD3
  054194h 4       EPIEN
  054198h 4       PIEN
  05419Ch 1x8     PAD4
  0541A4h 4       FNCTRL
  0541A8h 1x20    PAD5
  0541BCh 4       OTGREG
  0541C0h 1x12    PAD6
  0541CCh 4       DMASTART
  0541D0h 4       DMASTOP
  0541D4h 1x556   PAD7
  054400h 4       EP0DMAADDR
  054404h 1x28    PAD8
  054420h 4       EP1DMAADDR
  054424h 1x8     PAD9
  05442Ch 4       OUT1DMACTRL
  054430h 1x16    PAD10
  054440h 4       EP2DMAADDR
  054444h 1x8     PAD11
  05444Ch 4       OUT2DMACTRL
  054450h 1x16    PAD12
  054460h 4       EP3DMAADDR
  054464h 1x8     PAD13
  05446Ch 4       OUT3DMACTRL
  054470h 1x16    PAD14
  054480h 4       EP4DMAADDR
  054484h 1x8     PAD15
  05448Ch 4       OUT4DMACTRL
  054490h 1x16    PAD16
  0544A0h 4       EP5DMAADDR
  0544A4h 1x8     PAD17
  0544ACh 4       OUT5DMACTRL
  0544B0h 1x539472 PAD18            ;pad to BASE + 84000h
  0D8000h 4       USB_IP_BASE
```

rdma_reg.h (formerly at 00030100h in hw4.0) (hw6.0)

```
  054D00h 4       DMA_CONFIG
  054D04h 4       DMA_CONTROL
  054D08h 4       DMA_SRC
  054D0Ch 4       DMA_DEST
  054D10h 4       DMA_LENGTH
  054D14h 4       VMC_BASE
  054D18h 4       INDIRECT_REG
  054D1Ch 4       INDIRECT_RETURN
  054D20h 4x16    RDMA_REGION_(0..15)_
  054DA0h 4       DMA_STATUS
  054DA4h 4       DMA_INT_EN
```

athrI2cSlaveApbCsr.h (hw6.0)

```
  054E00h 4       I2CFIFOCONTROL
  054E04h 4       I2CFIFOREADPTR
  054E08h 4       I2CFIFOREADUPDATE
  054E0Ch 4       I2CFIFOREADBASEADDR
```

```
  054E10h 4       I2CFIFOWRITEPTR
  054E14h 4       I2CFIFOWRITEUPDATE
  054E18h 4       I2CFIFOWRITEBASEADDR
  054E1Ch 4       I2CMEMCONTROL
  054E20h 4       I2CMEMBASEADDR
  054E24h 4       I2CREGREADDATA
  054E28h 4       I2CREGWRITEDATA
  054E2Ch 4       I2CREGCONTROL
  054E30h 4       I2CCSRREADDATA
  054E34h 4       I2CCSRWRITEDATA
  054E38h 4       I2CCSRCONTROL
  054E3Ch 4       I2CFILTERSIZE
  054E40h 4       I2CADDR
  054E44h 4       I2CINT
  054E48h 4       I2CINTEN
  054E4Ch 4       I2CINTCSR
```

mbox_i2s_reg.h (hw6.0)

```
  055000h 4x1     MBOX_FIFO    ;<-- defined as array of ONE word (?)
  055004h 4       MBOX_FIFO_STATUS
  055008h 4       MBOX_DMA_POLICY
  05500Ch 4       MBOX0_DMA_RX_DESCRIPTOR_BASE
  055010h 4       MBOX0_DMA_RX_CONTROL
  055014h 4       MBOX0_DMA_TX_DESCRIPTOR_BASE
  055018h 4       MBOX0_DMA_TX_CONTROL
  05501Ch 4       MBOX_FRAME
  055020h 4       FIFO_TIMEOUT
  055024h 4       MBOX_INT_STATUS
  055028h 4       MBOX_INT_ENABLE
  05502Ch 4       MBOX_FIFO_RESET
  055030h 4       MBOX_DEBUG_CHAIN0
  055034h 4       MBOX_DEBUG_CHAIN1
  055038h 4       MBOX_DEBUG_CHAIN0_SIGNALS
  05503Ch 4       MBOX_DEBUG_CHAIN1_SIGNALS
```

map_rf_reg.h (hw6.0)

```
  xxx000h 4x256   RAM1[0..255]
  xxx400h 4x12    INT_PENDING[0..11]
  xxx430h 4       BB_WR_MASK_0                ;\
  xxx434h 4       BB_WR_MASK_1                ; BB Write Mask 0..3
  xxx438h 4       BB_WR_MASK_2                ;
  xxx43Ch 4       BB_WR_MASK_3                ;/
  xxx440h 4       RF_WR_MASK_0                ;\RF Write Mask 0..1
  xxx444h 4       RF_WR_MASK_1                ;/
  xxx448h 4       BB_RD_MASK_0                ;\
  xxx44Ch 4       BB_RD_MASK_1                ; BB Read Mask 0..3
  xxx450h 4       BB_RD_MASK_2                ;
  xxx454h 4       BB_RD_MASK_3                ;/
  xxx458h 4       RF_RD_MASK_0                ;\RF Read Mask 0..1
  xxx45Ch 4       RF_RD_MASK_1                ;/
  xxx460h 4       INT_SRC
  xxx464h 1x924   PAD__0
  xxx800h 4x128   RAM2[0..127]
```

odin_reg.h (hw6.0)

```
  xxx000h 4       PHY_CTRL0
  xxx004h 4       PHY_CTRL1
  xxx008h 4       PHY_CTRL2
  xxx00Ch 4       PHY_CTRL3
  xxx010h 4       PHY_CTRL4
  xxx014h 4       PHY_CTRL5
  xxx018h 4       PHY_CTRL6
  xxx01Ch 4       PHY_STATUS
```

DSi Atheros Wifi - Internal I/O - Unknown and Unused Registers (hw2)

hw4.0 would have the following extra registers

```
00D000h - DBG_UART Registers   ;\don't exist in hw2.0 ?
00E000h - UMBOX Registers      ;/
020000h - WMAC DMA and IRQ     ;\
020800h - WMAC QCU Queue       ; these MIGHT EXIST in hw2.0, too ?
021000h - WMAC DCU             ; (not defined in hw2.0 source code though)
029800h - BB/LC Registers      ;/
030100h - RDMA Registers       ;\don't exist in hw2.0 ?
031000h - EFUSE Registers      ;/
```

some partial hw2.0 memory dump...

```
000000 Deadc0de
004000 sth (01 00 00 00, 00 00 00 00)                       ;"RTC"
005000 Deadc0de
008000 sth (00 00 00 00, 00 00 00 00)                       ;"VMC?"
009000 Deadc0de           ;should contain UART etc. (maybe disabled?)
014000 sth (00 00 00 00, 00 00 00 00) (--crash-- at 0140cx)  ;"GPIO?"
015000 Deadc0de
018000 sth (00 01 0E 00, 00 01 0E 00)                       ;\MBOX
019000 sth (00 01 0E 00, 00 01 0E 00)                       ;/
01A000 sth (00 01 0E 00, 00 01 0E 00)                       ;\MBOX:HOST_IF?
01B000 sth (--crash-- at 01B00x)                            ;/
01C000 sth (00 00 14 00, D8 48 45 0E)                       ;-ANALOG?
01D000 Deadc0de
020000 sth (00's)                                           ;\DMA?
021000 sth (01 00 00 00, 02 00 00 00)                       ;/
022000 sth (00's)                                           ;\   ;\same as
023000 sth (01 00 00 00, 02 00 00 00)                       ;    ;/DMA?
024000 sth (00's)                                           ; ??
025000 sth (00's)                                           ;
026000 sth (00's)                                           ;
027000 sth (00's)                                           ;/
028000 sth (<--- mac addr ---> 86 38)           ;\MAC_PCU?     mac_pcu.h ?
029000 sth (14 E1 38 8A, 80 73 00 00)           ;/          ;\
02A000 sth (00's)                                           ; BB at 29800h?
02B000 sth (00's)                                           ;/
02C000 sth (00's)                                           ;-
02D000 sth (00's)                                           ;-
02E000 sth (00's)                                           ;-
02F000 sth (00's)                                           ;-
030000 Deadc0de              ;RDMA?? and (not?) EFUSE??
040000 Deadbeef        ;\
050000 Deadbeef        ;
060000 Deadbeef        ;
070000 Deadbeef        ;/
080000 004F1B74        ;\
090000 004F1B74        ;
0A0000 004F1B74        ; mirror of ROM word at [0F3FF8] (second-last-word)
0B0000 004F1B74        ;
0C0000 004F1B74        ;
0D0000 004F1B74        ;/
0E0000 sth (06 10 00 00, 21 22 22 22)   ;\80K ROM (14000h bytes)
0F0000 sth (00 00 05 60, FF DF FF FF)   ;/
0F4000 004F1B74        ;-mirror of ROM word at [0F3FF8] (second-last-word)
100000 sth (48 0F 8E 00, 70 14 50 00)   ;\
110000 sth                              ; 184K RAM (2E000h bytes)
120000 sth                              ;/
12E000 98A8A2AA        ;\
```

```
...                   ;
1FF000 98A8A2AA       ;/
200000 Deadbeef       ;\
300000 Deadbeef       ;/
00400000 looks like mirror of 000000
0041B000 looks like mirror of 01B000 --crash--
...      probably more mirrors...
FFC00000 looks like mirror of 000000
FFC1B000 looks like mirror of 01B000 --crash--
...      probably more mirrors till FFFFFFFF
```

some partial hw4.0 memory dump...

```
000000 zerofilled
004000 sth (01 00 00 00, 00 00 00 00)                      ;"RTC"
005000 zerofilled
040000 Deadbeef
080000 zerofilled
0E0000 06 10 00 00 21 22 22 22 00 00 00 E0 83 00 8E 00 ...   ;ROM?
109DC0 zerofilled                                            ;ROM?
10C000 14 19 52 00 ...                                       ;ROM?
114000 zerofilled                                            ;ROM?
120000 A0 B2 52 00     ;=52B2A0h (app_defined_area)   ;RAM
140000 zerofilled
200000 Deadbeef
400000..FFFFFFFF not checked (probably mirrors of above?)
```

DSi Atheros Wifi - Internal I/O - 004000h - RTC/Clock SOC (hw2/hw4/hw6)

ATH:004000h - WLAN/SOC_RESET_CONTROL ;hw2/hw4/hw6

```
0      SI0_RST
1      UART_RST
2      MBOX_RST
3      -
4      hw2/hw4: MAC_WARM_RST  ;-moved to 005000h.bit0 in hw6  ;\hw2/hw4 only
5      hw2/hw4: MAC_COLD_RST  ;-moved to 005000h.bit1 in hw6  ;/
6      CPU_WARM_RST
7      hw2/hw4: WARM_RST      ;-moved to 005000h.bit2 in hw6  ;-hw2/hw4 only
8      COLD_RST (0=no change, 1=reset)  ;-also in 005000h.bit3 in hw6
9      RST_OUT
10     hw2/hw4: VMC_REMAP_RESET   ;removed in hw6             ;-hw2/hw4 only
11     CPU_INIT_RESET
12     hw4: BB_WARM_RST       ;-moved to 005000h.bit4 in hw6  ;\hw4 only
13     hw4: BB_COLD_RST       ;-moved to 005000h.bit5 in hw6  ; (not hw2, and
14     hw4: DEBUG_UART_RST    ;-moved to bit16 in hw6         ;/moved in hw6)
12     hw6: MIT_ADAPTOR_RST                                ;\
13     hw6: MIT_REG_MAPPING_RST                            ;
14-15  hw6: -                                              ;
16     hw6: DEBUG_UART_RST ;<-- moved from old bit14       ;
17     hw6: UART2_RST                                      ;
18     hw6: CHECKSUM_ACC_RST                               ;
19     hw6: I2S_MBOX_RST                                   ;
20     hw6: I2S_RST                                        ;
21     hw6: GE0_RST                                        ;
22     hw6: MDIO_RST                                       ; hw6
23     hw6: MMAC_RST                                       ;
24     hw6: USB_RST                                        ;
25     hw6: USB_PHY_RST                                    ;
26     hw6: USB_PHY_ARST                                   ;
27     hw6: I2C_SLAVE_RST                                  ;
28     hw6: I2S_1_MBOX_RST                                 ;
```

```
  29     hw6: I2S_1_RST                                ;
  30     hw6: SPI2_RST                                 ;
  31     hw6: SDIO2_RST                                ;/
```

ATH:004008h - WLAN_TCXO_DETECT ;hw2/hw4 at this address
ATH:004004h - SOC_TCXO_DETECT ;hw6 at this address (unlike hw2/hw4)

```
  0      PRESENT
```

ATH:00400Ch - WLAN_XTAL_TEST ;hw2/hw4 at this address
ATH:004008h - SOC_XTAL_TEST ;hw6 at this address (unlike hw2/hw4)

```
  0      NOTCXODET
```

ATH:004020h - WLAN/SOC_CPU_CLOCK ;hw2/hw4/hw6

```
  0-1    STANDARD
```

ATH:004028h - WLAN/SOC_CLOCK_CONTROL ;hw2/hw4/hw6

```
  0      SI0_CLK
  1      hw2: UART_CLK      ;0=enable?    ;<-- hw2 only (removed in hw4/hw6)
  2      LF_CLK32
```

_____ Watchdog Timer _____

ATH:004030h - WLAN/SOC_WDT_CONTROL - Watchdog Timer Control ;hw2/hw4/hw6

```
  0-2    ACTION
```

ATH:004034h - WLAN/SOC_WDT_STATUS - Watchdog Interrupt Status ;hw2/hw4/hw6

```
  0      INTERRUPT
```

ATH:004038h - WLAN/SOC_WDT - Watchdog Timer Target ;hw2/hw4/hw6

```
  0-21   TARGET
```

ATH:00403Ch - WLAN/SOC_WDT_COUNT - Watchdog Timer Count ;hw2/hw4/hw6

```
  0-21   VALUE
```

ATH:004040h - WLAN/SOC_WDT_RESET - Watchdog Timer Reset ;hw2/hw4/hw6

```
  0      VALUE
```

_____ Interrupt Status _____

ATH:004044h - WLAN/SOC_INT_STATUS ;hw2/hw4/hw6

```
  0      WDT_INT        ;-Watchdog Timer
  1      ERROR
  2      UART           ;-Serial UART
  3      GPIO           ;-GPIO
  4      SI             ;-Serial I2C/SPI
  5      KEYPAD
  6      LF_TIMER0      ;\
  7      LF_TIMER1      ; Low-Freq Timer 0..3
  8      LF_TIMER2      ; and
  9      LF_TIMER3      ; High-Freq Timer
  10     HF_TIMER       ;/
  11     RTC_ALARM      ;-Real-Time Clock Alarm
  12     MAILBOX
  13     MAC            ;-maybe this is "MAC's INTA#" (see WMAC IRQ) ?
  14     RTC_POWER
  15     hw4/hw6: BTCOEX   ;Bluetooth Coex      ;\
  16     hw4/hw6: RDMA                          ;
  17     hw4/hw6: GENERIC_MBOX  (aka GMBOX)     ; hw4/hw6
  18     hw4/hw6: UART_MBOX                     ;
  19     hw4/hw6: EFUSE_OVERWRITE               ;
```

```
 20     hw4/hw6: THERM                          ;
 21     hw4/hw6: HCI_UART                       ;/
 22     hw6: MODE_SWITCH                        ;\
 23     hw6: RF_SLEEP_RISING                    ;
 24     hw6: BBP_SLEEP_RISING                   ;
 25     hw6: FLIGHT_MODE                        ;
 26     hw6: MIT_REG_ACCESS                     ; hw6 only
 27     hw6: MMAC                               ;
 28     hw6: USBIP                              ;
 29     hw6: USBDMA                             ;
 30     hw6: SDIO2_MBOX                         ;
 31     hw6: STE_MBOX                           ;/
```

ATH:0040CCh - WLAN/SOC_INT_STATUS1 ;hw6

```
 0      MAC_1                                   ;\
 1      MAC_2                                   ;
 2      MAC_3                                   ;
 3      MAC_4                                   ; hw6 only
 4      CKSUM                                   ; (additional bits, extending
 5      I2C_S                                   ; the bits in port 004044h)
 6      GMAC                                    ;
 7      MDIO                                    ;
 8      I2S                                     ;
 9      I2S_1                                   ;/
```

_____ Low-Freq Timer 0-3 and High-Freq Timer _____

ATH:004048h - WLAN/SOC_LF_TIMER0 ;hw2/hw4/hw6
ATH:004058h - WLAN/SOC_LF_TIMER1 ;hw2/hw4/hw6
ATH:004068h - WLAN/SOC_LF_TIMER2 ;hw2/hw4/hw6
ATH:004078h - WLAN/SOC_LF_TIMER3 ;hw2/hw4/hw6
ATH:004088h - WLAN/SOC_HF_TIMER ;hw2/hw4/hw6

```
For LF Timer 0-3:
 0-31   TARGET
For HF Timer:
 12-31  TARGET      ;<-- not bit0-31 for HF timer
```

ATH:00404Ch - WLAN/SOC_LF_TIMER_COUNT0 ;hw2/hw4/hw6
ATH:00405Ch - WLAN/SOC_LF_TIMER_COUNT1 ;hw2/hw4/hw6
ATH:00406Ch - WLAN/SOC_LF_TIMER_COUNT2 ;hw2/hw4/hw6
ATH:00407Ch - WLAN/SOC_LF_TIMER_COUNT3 ;hw2/hw4/hw6
ATH:00408Ch - WLAN/SOC_HF_TIMER_COUNT ;hw2/hw4/hw6

```
For LF Timer 0-3:
 0-31   VALUE
For HF Timer:
 12-31  VALUE       ;<-- not bit0-31 for HF timer
```

ATH:004090h - WLAN/SOC_HF_LF_COUNT ;hw2/hw4/hw6

```
 0-31   VALUE       ;<-- extra for HF timer
```

ATH:004050h - WLAN/SOC_LF_TIMER_CONTROL0 ;hw2/hw4/hw6
ATH:004060h - WLAN/SOC_LF_TIMER_CONTROL1 ;hw2/hw4/hw6
ATH:004070h - WLAN/SOC_LF_TIMER_CONTROL2 ;hw2/hw4/hw6
ATH:004080h - WLAN/SOC_LF_TIMER_CONTROL3 ;hw2/hw4/hw6
ATH:004094h - WLAN/SOC_HF_TIMER_CONTROL ;hw2/hw4/hw6

```
For both LF and HF:
 0      RESET
 1      AUTO_RESTART
For LF Timer 0-3:
 2      ENABLE
```

```
For HF Timer:
  2      ON          ;<-- extra bit for HF timer
  3      ENABLE      ;<-- moved to bit3
```

ATH:004054h - WLAN/SOC_LF_TIMER_STATUS0 ;hw2/hw4/hw6
ATH:004064h - WLAN/SOC_LF_TIMER_STATUS1 ;hw2/hw4/hw6
ATH:004074h - WLAN/SOC_LF_TIMER_STATUS2 ;hw2/hw4/hw6
ATH:004084h - WLAN/SOC_LF_TIMER_STATUS3 ;hw2/hw4/hw6
ATH:004098h - WLAN/SOC_HF_TIMER_STATUS ;hw2/hw4/hw6

```
  0      INTERRUPT
```

Real-Time Clock

The DSi does have a battery-backed RTC accessed via Port 4000138h on ARM7 side. Below SOC_RTC would offer a second RTC, but it isn't actually used in DSi (the SOC_RTC registers aren't battery-backed, and the DSi firmware isn't initializing them from the ARM7 time either).

ATH:00409Ch - WLAN/SOC_RTC_CONTROL ;hw2/hw4/hw6

```
  0      LOAD_ALARM
  1      LOAD_RTC
  2      ENABLE
```

ATH:0040A0h - WLAN/SOC_RTC_TIME ;hw2/hw4/hw6
ATH:0040A8h - WLAN/SOC_RTC_SET_TIME ;hw2/hw4/hw6

```
  0-6    SECOND
  8-14   MINUTE
  16-21  HOUR
  24-26  WEEK_DAY
```

ATH:0040A4h - WLAN/SOC_RTC_DATE ;hw2/hw4/hw6
ATH:0040ACh - WLAN/SOC_RTC_SET_DATE ;hw2/hw4/hw6

```
  0-5    MONTH_DAY
  8-12   MONTH
  16-23  YEAR
```

ATH:0040B0h - WLAN/SOC_RTC_SET_ALARM ;hw2/hw4/hw6

```
  0-6    SECOND
  8-14   MINUTE
  16-21  HOUR
```

ATH:0040B4h - WLAN/SOC_RTC_CONFIG ;hw2/hw4/hw6

```
  0      DSE
  1      TWELVE_HOUR
  2      BCD
```

ATH:0040B8h - WLAN/SOC_RTC_ALARM_STATUS ;hw2/hw4/hw6

```
  0      INTERRUPT
  1      ENABLE
```

Chip ID

ATH:0040ECh - WLAN_CHIP_ID ;hw2/hw4 - single ID in hw2/hw4
ATH:0040ECh - LEGACY_SOC_CHIP_ID ;hw6 - first/legacy/old ID in hw6
ATH:0040F0h - SOC_CHIP_ID ;hw6 - second/actual/new ID in hw6

```
  0-3    VERSION_ID (4bit, usually 0 or 1)
  4-15   CONFIG_ID (12bit, usually 0)
  16-31  DEVICE_ID (16bit, usually xx00h or xx01h for AR60xx, eg. 0D00h=AR6013)
```

The DSi Wifi Firmware file contains a list of supported ID(s) for each chip:

```
  AR6002: 02010001h or 02000001h (the latter one being actually used in DSi)
```

```
  AR6013: 0D000000h or 0D000001h (unknown which one is actually used in DSi)
```
For 3DS:
```
  AR6014: returns 0D000001h=[0040ECh]
```
Whereas, the DSi is using that value only for verification, not for actually detecting the installed chip type (that is done by a byte in SPI FLASH).
Note: There's also a "BB_D2_CHIP_ID" register in the Baseband section.

_____ Misc _____

ATH:0040BCh - WLAN/SOC_UART_WAKEUP ;hw2/hw4/hw6
```
  0      ENABLE
```

ATH:0040C0h - WLAN/SOC_RESET_CAUSE ;hw2/hw4/hw6
```
  0-2    Cause of most recent Reset event (LAST)
```
Possible values are (according to AR6001 datasheet - might be different in hw2/hw4/hw6):
```
  00h = SYS_RST_L pin was asserted
  01h = Host wrote to the SDIO reset register
  02h = Software wrote RTC_CONTROL_COLD_RST register (aka COLD_RST bit?)
  03h = Software wrote RTC_CONTROL_WARM_RST register (aka WARM_RST bit?)
  04h = Software wrote RTC_CONTROL_CPU_RST register  (aka CPU_WARM_RST bit?)
  05h = Watchdog Timer has expired
  06h..07h = Reserved
```

ATH:0040C4h - WLAN/SOC_SYSTEM_SLEEP ;hw2/hw4/hw6
```
  0      DISABLE
  1      LIGHT
  2      MAC_IF
  3      MBOX
  4      HOST_IF
  5      hw6: MCI                              ;-hw6 only
```

ATH:0040C8h - WLAN/SOC_SDIO_WRAPPER ;hw2/hw4/hw6
```
  0      ON
  1      SOC_ON
  2      WAKEUP
  3      SLEEP
```

ATH:004110h - WLAN/SOC_POWER_REG - located here in hw2
ATH:00410Ch - WLAN/SOC_POWER_REG - located here in hw4/hw6
```
  0      POWER_EN
  1      WLAN_PWD_EN
  2      hw2: WLAN_SCALE_EN            ;\
  2      hw4: WLAN_ISO_EN              ; hw2/hw4 (renamed from hw2:SCALE
  3      hw2: SOC_SCALE_EN             ; to hw4:ISO, and removed in hw6)
  3      hw4: SOC_ISO_EN               ;/
  4      RADIO_PWD_EN
  5      hw2/hw4: WLAN_ISO_CNTL        ;\hw2/hw4 (removed? in hw6)
  6      hw2/hw4: WLAN_ISO_DIS         ;/
  7      CPU_INT_ENABLE
  8-11   hw2/hw4: VLVL                 ;-hw2/hw4 (removed? in hw6)
  12     hw4/hw6: WLAN_MAC_PWD_EN      ;\
  13     hw4/hw6: WLAN_BB_PWD_EN       ; hw4/hw6
  14     hw4/hw6: DEBUG_EN             ;/
  15     hw4: SLEEP_MAKE_N_BREAK_EN    ;\hw4+hw6, but changed/renamed?
  15     hw6: DEEP_SLEEP_EN            ;/
  16     hw6: DISCON_MODE_EN           ;\hw6
  17     hw6: SWREG_VS_EN              ;/
```

ATH:004114h - WLAN/SOC_CORE_CLK_CTRL - located here in hw2
ATH:004110h - WLAN/SOC_CORE_CLK_CTRL - located here in hw4/hw6
```
  0-2    DIV
```

ATH:0041C0h - WLAN/SOC_GPIO_WAKEUP_CONTROL - located here in hw2
ATH:004114h - WLAN/SOC_GPIO_WAKEUP_CONTROL - located here in hw4/hw6

```
0       ENABLE
```

ATH:004214h - SLEEP_RETENTION ;hw4/hw6

```
0       ENABLE
1       MODE
2-9     TIME
```

_____ LPO _____

ATH:0040D4h - WLAN/SOC_LPO_CAL_TIME ;hw2/hw4/hw6

```
0-13    LENGTH
```

ATH:0040D8h - WLAN/SOC_LPO_INIT_DIVIDEND_INT ;hw2/hw4/hw6

```
0-23    VALUE
```

ATH:0040DCh - WLAN/SOC_LPO_INIT_DIVIDEND_FRACTION ;hw2/hw4/hw6

```
0-10    VALUE
```

ATH:0040E0h - WLAN/SOC_LPO_CAL ;hw2/hw4/hw6

```
0-19    COUNT
20      ENABLE
```

ATH:0040E4h - WLAN/SOC_LPO_CAL_TEST_CONTROL ;hw2/hw4/hw6

```
0-4     hw2/hw4: RTC_CYCLES (5bit)      ;\hw2/hw4 (5bit)
5       hw2/hw4: ENABLE                 ;/
0-15    hw6:     RTC_CYCLES (16bit)     ;\hw6 (expanded to 16bit)
16      hw6:     ENABLE                 ;/(and moved enable flag)
```

ATH:0040E8h - WLAN/SOC_LPO_CAL_TEST_STATUS ;hw2/hw4/hw6

```
0-15    COUNT
16      READY
```

_____ below in hw4/hw6 only _____

ATH:004284h - LP_PERF_COUNTER ;hw4/hw6

```
0       EN
```

ATH:004288h - LP_PERF_LIGHT_SLEEP ;hw4/hw6
ATH:00428Ch - LP_PERF_DEEP_SLEEP ;hw4/hw6
ATH:004290h - LP_PERF_ON - hw4 only

```
0-31    CNT
```

_____ MISC _____

ATH:0042A8h - CHIP_MODE ;hw4/hw6

```
0-1     BIT
```

ATH:0042ACh - CLK_REQ_FALL_EDGE ;hw4/hw6

```
0-7     DELAY
31      EN
```

_____ OTP _____

ATH:0042B0h - OTP ;hw4/hw6

```
0       VDD12_EN
```

```
  1       LDO25_EN
```

ATH:0042B4h - OTP_STATUS ;hw4/hw6

```
  0       VDD12_EN_READY
  1       LDO25_EN_READY

 ____________________________________ PMU _____________________________________
```

ATH:0042B8h - PMU ;hw4/hw6

```
  0-1     REG_WAKEUP_TIME_SEL
```

ATH:0042C0h..42C4h - PMU_CONFIG[0..1] ;hw4

```
  0-15    VALUE ... whatever... (2x16bit)
  16-31   -
```

Maybe this arrary entries are equivalent/similar to the PMU_CONFIG and PMU_PAREG registers in hw6? The hw4 source code is claiming the array entries to be 16bit wide each, which doesn't match up with the two 5bit/3bit registers on hw6 though.

ATH:0042BCh - PMU_CONFIG ;hw6

```
  0-4     VALUE
```

Maybe equivalent/similar to one of the PMU_CONFIG[0..1] entries in hw4?

ATH:0042C0h - PMU_PAREG ;hw6

```
  0-2     LVL_CTR
```

Maybe equivalent/similar to one of the PMU_CONFIG[0..1] entries in hw4?

ATH:0042C8h - PMU_BYPASS ;hw4 at this address
ATH:0042C4h - PMU_BYPASS ;hw6 at this address (unlike hw4)

```
  0       hw4/hw6: PAREG                          ;-hw4/hw6 ;\
  1       hw4: DREG       ;-removed in hw6        ;\hw4     ; hw4/hw6 only
  2       hw4: SWREG      ;-moved to bit1 in hw6  ;/        ; (not hw2)
  1       hw6: SWREG      ;-formerly in bit2      ;-hw6     ;/

 ___________________________________ THERM CTRL ___________________________________
```

ATH:0042DCh - THERM_CTRL1 ;hw4/hw6

```
  0       INT_STATUS
  1       INT_EN
  2       MEASURE
  3-4     TYPE
  5-11    WIDTH
  12-15   WIDTH_ARBITOR
  16      BYPASS
```

ATH:0042E0h - THERM_CTRL2 ;hw4/hw6

```
  0-7     LOW
  8-15    HIGH
  16-23   SAMPLE
  24      ADC_ON
  25      ADC_OFF
```

ATH:0042E4h - THERM_CTRL3 ;hw4/hw6

```
  0-7     ADC_OFFSET
  8-16    ADC_GAIN

 ______________________________ below in hw6 only ______________________________
```

ATH:0042E8h - LISTEN_MODE1 ;hw6

```
0       ENABLE
1       CLOCK_GATE
2       TIMER_OVERFLOW_WAKE
3-18    TIMER_THRESH_WAKE
19      TIMER_CLEAR
```

ATH:0042ECh - LISTEN_MODE2 ;hw6

```
0-15    TIMER_TRIGGER_WAKE
```

ATH:0042F0h - AUDIO_PLL_CONFIG ;hw6

```
0-3     REFDIV
4       BYPASS
5       PLLPWD
7-9     POSTPLLDIV
12-14   EXT_DIV
31      UPDATING
```

ATH:0042F4h - AUDIO_PLL_MODULATION ;hw6

```
0       START
1-6     TGT_DIV_INT
11-28   TGT_DIV_FRAC
```

ATH:0042F8h - AUDIO_PLL_MOD_STEP ;hw6

```
0-3     UPDATE_CNT
4-13    INT
14-31   FRAC
```

ATH:0042FCh - CURRENT_AUDIO_PLL_MODULATION ;hw6

```
1-6     INT
10-27   FRAC
```

ATH:004300h - ETH_PLL_CONFIG ;hw6

```
0-4     REFDIV
5       BYPASS
6       PLLPWD
7-9     OUTDIV
12-17   INT
18-27   FRAC
28      RANGE
29      GE0
30      GE0_MASTER
```

ATH:004304h - CPU_PLL_CONFIG ;hw6

```
0-4     REFDIV
6       PLLPWD
7-9     OUTDIV
12-17   INT
20-25   FRAC
28      RANGE
```

ATH:004308h - BB_PLL_CONFIG ;hw6

```
0-17    FRAC
```

ATH:00430Ch - ETH_XMII ;hw6

```
0-7     PHASE0_COUNT
8-15    PHASE1_COUNT
16-23   OFFSET_COUNT
24      OFFSET_PHASE
25      GIGE
26-27   TX_DELAY
28-29   RX_DELAY
```

```
30      GIGE_QUAD
31      TX_INVERT
```

ATH:004310h - USB_PHY_CONFIG ;hw6
```
0       HOSTMODE
1       PLL_PWD
2       TESTMODE
3       REFDIV
4-7     REFCLK_SEL
```

ATH:004314h - MITSUMI_INT_CONTROL_REG ;Mitsumi Interrupt Enable ;hw6
ATH:004318h - MITSUMI_INT_STATUS_REG ;Mitsumi Interrupt Status ;hw6
```
0       MODE_SWITCH
1       RF_SLEEP
2       BBP_SLEEP
3       FLIGHT_MODE
```
Maybe this is related to the Nintendo DSi's backwards compatibilty mode (in which, the Atheros chip is simulating a Mitsumi BB/RF chip for use with older NDS games)? Or maybe it's unrelated to that (DSi/3DS don't seem to have hw6 at all).

ATH:00431Ch - CURRENT_WORKING_MODE ;hw6
```
0       VALUE
1       NOT_FIRST_MIT_MODE
2       MIT_REG_WR_TRIGGER_EN
5       MIT_FORCE_ACTIVE_ON
```

ATH:004320h - RTC_SLEEP_COUNT ;hw6
```
0-5     THRESHOLD
```

ATH:004324h - MIT2_VAP ;hw6
```
0       MODE
```

ATH:004328h - SECOND_HOST_INFT ;hw6
```
0       SDIO_MODE
```

ATH:00432Ch - SDIO_HOST ;hw6
```
0       RESET
```

ATH:004330h - ENTERPRISE_CONFIG ;hw6
```
0       LOCATION_DISABLE
1       LOOPBACK_DISABLE
2       MIN_PKT_SIZE_DISABLE
3       TXBF_DISABLE
4       CH_10MHZ_DISABLE
5       CH_5MHZ_DISABLE
6       CHAIN1_DISABLE
7       DUAL_BAND_DISABLE
8       GREEN_TX_DISABLE
9       LDPC_DISABLE
10      STBC_DISABLE
11      SWCOM_IDLE_MODE
12      TPC_LOWER_PERFORMANCE
```

ATH:004334h - RTC_DEBUG_BUS ;hw6
```
0       SEL
```

ATH:004338h - RTC_EXT_CLK_BUF ;hw6
```
0       EN
```

DSi Atheros Wifi - Internal I/O - 00x000h - RTC/Clock WLAN (hw2/hw4/hw6)

ATH:004004h/004004h/005004h - WLAN_XTAL_CONTROL ;hw2/hw4/hw6

```
0      TCXO
```

ATH:004010h/004010h/005010h - WLAN_QUADRATURE ;hw2/hw4/hw6

```
0-1    hw2/hw4: DAC (2bit)   ;\expanded DAC from 2bit (hw2/hw4) to 3bit (hw6)
0-2    hw6:     DAC (3bit)   ; (and removed SEL bit in hw6)
2      hw2/hw4: SEL          ;/
4-5    hw2:     ADC (2bit)   ;\expanded ADC from 2bit (hw2) to 4bit (hw4)
4-7    hw4/hw6: ADC (4bit)   ;/
```

ATH:004014h/004014h/005014h - WLAN_PLL_CONTROL ;hw2/hw4/hw6

```
0-9    DIV
12-15  hw2/hw4: REFDIV (4bit)
10-13  hw6:     REFDIV (4bit, now here)
14-15  hw6:     CLK_SEL ;<-- maybe replaces removed "SEL" in WLAN_QUADRATURE?
16     BYPASS
17     UPDATING                      (R)
18     NOPWD
19     MAC_OVERRIDE
20     DIG_TEST_CLK
```

ATH:004018h/004018h/005018h - WLAN_PLL_SETTLE ;hw2/hw4/hw6

```
0-11   hw2/hw4: TIME (12bit)         ;\decreased from 12bit to 11bit in hw6
0-10   hw6:     TIME (11bit)         ;/
```

ATH:00401Ch/00401Ch/00501Ch - WLAN_XTAL_SETTLE ;hw2/hw4/hw6

```
0-7    hw2/hw4: TIME (8bit)          ;\decreased from 8bit to 7bit in hw6
0-6    hw6:     TIME (7bit)          ;/
```

ATH:004020h/004020h/005020h - WLAN_CLOCK_OUT ;hw2/hw4/hw6

```
0-3    hw2/hw4: SELECT (4bit)        ;\raised from 4bit to 5bit in hw6,
0-4    hw6:     SELECT (5bit)        ; and added new DELAY field in hw6
5-7    hw6:     DELAY (3bit, new)    ;/
```

ATH:00402Ch/00402Ch/005024h - WLAN_BIAS_OVERRIDE ;hw2/hw4/hw6

```
0      ON
```

ATH:0040CCh/0040CCh/005030h - WLAN_MAC_SLEEP_CONTROL ;hw2/hw4/hw6

```
0-1    hw2/hw4: ENABLE               ;\
0      hw6:     ENABLE               ; reduced from 2bit to 1bit in hw6
1      hw6:     RESERVED             ;/
2      hw6:     HSEL_WMAC_ENABLE     ;-new in hw6
```

ATH:0040D0h/0040D0h/005034h - WLAN_KEEP_AWAKE ;hw2/hw4/hw6

```
0-7    COUNT
```

ATH:0040F0h/0040F0h/005038h - WLAN_DERIVED_RTC_CLK ;hw2/hw4/hw6

```
1-15   PERIOD
16-17  hw2/hw4: FORCE                ;-hw2/hw4 only (removed in hw6)
18     EXTERNAL_DETECT           (R)
20     EXTERNAL_DETECT_EN
```

_____ SLP or SLOP or SLEEP or so _____

ATH:0040F4h/0040F4h/00503Ch - MAC_PCU_SLP32_MODE ;hw2/hw4/hw6

ATH:028244h (mirror of 0040F4h) - MAC_PCU_REG_SLP32_MODE (ini:10F424h) ;hw2

```
  0-19   HALF_CLK_LATENCY
  20     ENABLE    ;<-- see hw2 note    ;-hw2/hw4/hw6 (on hw2 in mirror only)
  21     hw2:     TSF_WRITE_PENDING     ;\changed/renamed in hw2/hw4
  21     hw4/hw6: TSF_WRITE_STATUS  (R) ;/
  22     hw4/hw6: DISABLE_32KHZ         ;\
  23     hw4/hw6: FORCE_BIAS_BLOCK_ON   ; hw4/hw6 only (unspecified in hw2)
  24     hw4/hw6: TSF2_WRITE_STATUS (R) ;/
```

In hw2, this register appears to be mirrored to 0040F4h (without ENABLE flag in bit20) and 028244h (bit ENABLED flag in bit20).

ATH:0040F8h/0040F8h/005040h - MAC_PCU_SLP32_WAKE ;hw2/hw4/hw6
ATH:028248h (mirror of 0040F8h) - MAC_PCU_REG_SLP32_WAKE (ini:07EFh) ;hw2

```
  0-15   XTL_TIME
```

ATH:0040FCh/0040FCh/005044h - MAC_PCU_SLP32_INC ;hw2/hw4/hw6
ATH:02824Ch (mirror of 0040FCh) - MAC_PCU_REG_SLP32_TSF_INC (ini:1E848h) ;hw2

```
  0-19   TSF_INC
```

ATH:004100h/004100h/005048h - MAC_PCU_SLP_MIB1 ;hw2/hw4/hw6
ATH:028250h (mirror of 004100h) - MAC_PCU_REG_SLPMIB1 ;hw2

```
  0-31   SLEEP_CNT
```

ATH:004104h/004104h/00504Ch - MAC_PCU_SLP_MIB2 ;hw2/hw4/hw6
ATH:028254h (mirror of 004104h) - MAC_PCU_REG_SLPMIB2 ;hw2

```
  0-31   CYCLE_CNT
```

ATH:004108h/004108h/005050h - MAC_PCU_SLP_MIB3 ;hw2/hw4/hw6
ATH:028258h (mirror of 004108h) - MAC_PCU_REG_SLPMIB3 ;hw2

```
  0      CLR_CNT
  1      PENDING                    (R)
```

In hw2, this register appears to be mirrored to 004108h (with bit1 called "PENDING") and 028258h (with bit1 called "PEND" for whatever reason).

ATH:0280D4h/004204h/0050C0h - MAC_PCU_SLP1 ;hw2/hw4/hw6

```
  0-18   hw2: outcommented: NEXT_DTIM)   (hw2: ini:2AAAAh) ;\outcommented
  20     hw2: outcommented: ENH_SLEEP_ENABLE) (hw2: ini:1) ;/
  0-4    hw2/hw6: CAB_TIMEOUT_EXT             (hw2: ini:0) ;-hw2/hw6
  0-15   hw4: CAB_TIMEOUT                                  ;-hw4
  19     ASSUME_DTIM                          (hw2: ini:0) ;-hw2/hw4/hw6
  20     hw6: BUG_59985_FIX_ENABLE                         ;-hw6
  21-31  hw2/hw6: CAB_TIMEOUT                 (hw2: ini:5) ;-hw2/hw6
```

ATH:0280D8h/004208h/0050C4h - MAC_PCU_SLP2 ;hw2/hw4/hw6

```
  0-18   hw2: outcommented: NEXT_TIM  (hw2: ini:55555h) ;-outcommented
  0-15   hw4: BEACON_TIMEOUT                            ;-hw4
  0-4    hw2/hw6: BEACON_TIMEOUT_EXT  (hw2: ini:0)      ;\hw2/hw6
  21-31  hw2/hw6: BEACON_TIMEOUT      (hw2: ini:2)      ;/
```

ATH:0280DCh - (outcommented) ;aka MAC_PCU_REG_SLP3 ;hw2 (but outcommented)

```
  0-15   hw2: outcommented: TIM_PERIOD  (hw2: ini:2)    ;\outcommented
  16-31  hw2: outcommented: DTIM_PERIOD (hw2: ini:3)    ;/
```

ATH:028260h/0050C8h - MAC_PCU_SLP3 ;hw2/hw6

```
  0-15   hw2/hw6: CAB_AWAKE_DUR        (hw2: ini:0005h) ;\hw2/hw6
  16     hw2/hw6: CAB_AWAKE_ENABLE     (hw2: ini:0)     ;/
```

ATH:0050CCh - MAC_PCU_SLP4 ;hw6

```
  0-15   hw6: BEACON2_TIMEOUT
```

```
  16-30  hw6: CAB2_TIMEOUT
  31     hw6: ASSUME_DTIM2
```

```
 _____________________________ Generic Timers _____________________________
```

ATH:028200h..02821Ch - MAC_PCU_REG_GNRCTMR_N[0..7] ;hw2
ATH:028220h..02823Ch - MAC_PCU_REG_GNRCTMR_P[0..7] ;hw2
ATH:004140h..417Ch/005064h..50A0h - MAC_PCU_GENERIC_TIMERS[0..15] ;hw4/hw6
ATH:0041C0h..41FCh/0050DCh..5118h - MAC_PCU_GENERIC_TIMERS2[0..15] ;hw4/hw6

```
  0-31   hw2, for "GNRCTMR_N" entries: GNRCTMR_N (32bit)
  0-27   hw2, for "GNRCTMR_P" entries: GNRCTMR_P (only 28bit here)
  0-31   hw4/hw6: DATA (32bit)
```

Unknown how that stuff is related...

```
  - hw2   has "8xTMR_N"  plus "8xTMR_P"
  - hw4/6 has "16xTIMER" plus "16xTIMER2"
```

maybe TIMER and TIMER2 are equivalent to TMR_N and TMR_P, or maybe TIMER is meant to contain "eight N+P pairs" (and TIMER2 another eight pairs), or maybe the "N" and "P" stuff was completely dropped in hw4/hw6. The ENABLE bits are also weird:

```
  - hw2 has 1x8 ENABLE bits  (for 8+8 timer entries)
  - hw4 has 2x16 ENABLE bits (for 16+16 timer entries)
  - hw6 has 2x8 ENABLE bits  (for 16+16 timer entries, too)
```

Note that hw2 has the generic timers in the WMAC PCU area at 028xxxh.

ATH:028240h/004180h/0050BCh - MAC_PCU_GENERIC_TIMERS_MODE ;hw2/hw4/hw6

```
  0-15   hw4: ENABLE         (16bit)            ;\hw4 (the other bits are
  16-31  hw4: -                                 ;/moved to "MODE3" in hw4)
  0-7    hw6: ENABLE         (8bit)             ;\
  8-10   hw6: OVERFLOW_INDEX (3bit)        (R)  ; hw2/hw6
  11     hw6: -                                 ;
  12-31  hw6: THRESH         (20bit)            ;/
```

ATH:004200h/005134h - MAC_PCU_GENERIC_TIMERS_MODE2 ;hw4/hw6 (not hw2)

```
  0-15   hw4: ENABLE          (16bit)           ;-hw4
  0-7    hw6: ENABLE          (8bit)            ;\
  8-11   hw6: OVERFLOW_INDEX  (4bit)        (R) ; hw6
  12-15  hw6: OVERFLOW_INDEX2 (4bit)        (R) ;/
```

ATH:0042D4h - MAC_PCU_GENERIC_TIMERS_MODE3 ;hw4 only

```
  0-19   hw4: THRESH                    ;\hw4 only (in hw2/hw6 this stuff is
  24-27  hw4: OVERFLOW_INDEX            ;/located in "MODE" instead of "MODE3")
```

ATH:005150h - MAC_PCU_GENERIC_TIMERS_TSF_SEL ;hw6 only

```
  0-15   VALUE
```

```
 ____________________________ below in hw4/hw6 only ____________________________
```

ATH:004118h/005148h - WLAN_HT (aka HT) ;hw4/hw6

```
  0      MODE
```

ATH:00411Ch/005054h - MAC_PCU_TSF_L32 ;hw4/hw6 (hw2: see REG_TSF_L32)
ATH:004120h/005058h - MAC_PCU_TSF_U32 ;hw4/hw6

```
  0-63   VALUE
```

ATH:0042CCh/0050D4h - MAC_PCU_TSF2_L32 ;hw4/hw6
ATH:0042D0h/0050D8h - MAC_PCU_TSF2_U32 ;hw4/hw6

```
  0-63   VALUE
```

ATH:00505Ch - MAC_PCU_WBTIMER_0 ;hw6

```
  0      ENABLE                             ;-hw6 only
```

ATH:004124h/005060h - MAC_PCU_WBTIMER_1 (aka MAC_PCU_WBTIMER) ;hw4/hw6
```
  0-31   VALUE
```

ATH:00420Ch/0050D0h - MAC_PCU_RESET_TSF ;hw4/hw6
```
  24     ONE_SHOT    ;aka "one shot RESET_TSF"   ;<-- see "REG_BEACON" in hw2
  25     ONE_SHOT2
```

ATH:00427Ch/005154h - MAC_PCU_BMISS_TIMEOUT ;hw4/hw6
ATH:005158h - MAC_PCU_BMISS2_TIMEOUT ;hw6 only
```
  0-23   VALUE  (aka BMISS_TIMEOUT)
  24     ENABLE (aka BMISS_TIMEOUT_ENABLE)
```

_____ below in hw2 only _____

ATH:00410Ch - MAC_PCU_SLP_BEACON - hw2 only (removed or renamed in hw4)
ATH:028264h (mirror of 00410Ch) - MAC_PCU_REG_SLP5 (ini: 0FFFFFFh) ;hw2
```
  0-23   hw2: BMISS_TIMEOUT                  ;\hw2 only
  24     hw2: BMISS_TIMEOUT_ENABLE           ;/
```
See also: - MAC_PCU_BMISS_TIMEOUT <--- is that equivalent on hw4/hw6!?!

ATH:004120h..413Ch - SDIO_SETUP_CIRCUIT[8] - hw2 only (removed in hw4/hw6)
ATH:004160h..417Ch - CPU_SETUP_CIRCUIT[8] - hw2 only (removed in hw4/hw6)
ATH:0041A0h..41BCh - BB_SETUP_CIRCUIT[8] - hw2 only (removed in hw4/hw6)
```
  0-7    hw2: VECTOR
```

ATH:004140h - SDIO_SETUP_CONFIG - hw2 only (removed in hw4/hw6)
ATH:004144h - CPU_SETUP_CONFIG - hw2 only (removed in hw4/hw6)
ATH:004180h - BB_SETUP_CONFIG - hw2 only (removed in hw4/hw6)
```
  0      hw2: CLEAR
  1      hw2: ENABLE
```

_____ below in hw4 only _____

ATH:004210h - MAC_PCU_TSF_ADD_PLL ;hw4
```
  0-7    hw4: VALUE
```

ATH:004280h - MAC_PCU_CAB_AWAKE ;hw4
```
  0-15   hw4: DURATION
  16     hw4: ENABLE
```

ATH:0042D8h - MAC_PCU_DIRECT_CONNECT ;hw4
```
  0      hw4: AP_STA_ENABLE
  1      hw4: AP_TSF_1_2_SEL
  2      hw4: STA_TSF_1_2_SEL
```

ATH:004218h..004278h - Bluetooth Coex related ;hw4
ATH:004294h..0042A4h - Bluetooth Coex related, too? ;hw4
These hw4 ports have been moved to 007xxxh in hw6. See Bluetooth Coex chapter for details.

_____ below in hw6 only _____

ATH:005000h - WLAN_RESET_CONTROL ;hw6
```
  0      MAC_WARM_RST           ;-moved from 004000h.bit4   ;\
  1      MAC_COLD_RST           ;-moved from 004000h.bit5   ;
  2      WARM_RST               ;-moved from 004000h.bit7   ;
  3      COLD_RST               ;-also in 004000h.bit8      ;
  4      BB_WARM_RST            ;-moved from 004000h.bit12  ;
```

```
  5      BB_COLD_RST            ;-moved from 004000h.bit13  ;/
  6      RADIO_SRESET           ;-new hw6 bit               ;\
  7      MCI_RESET              ;-new hw6 bit               ;/
```

ATH:005008h - WLAN_REG_CONTROL0 ;hw6

```
  0-31   SWREG_BITS
```

ATH:00500Ch - WLAN_REG_CONTROL1 ;hw6

```
  0      SWREG_PROGRAM
  1-2    OTPREG_LVL
```

ATH:005028h - WLAN_RESET_CAUSE ;hw6 ;<--- mirror of SOC_RESET_CAUSE?

```
  0-2    LAST                        (R)
```

ATH:00502Ch - WLAN_SYSTEM_SLEEP ;hw6 ;<--partial mirror of SOC_SYSTEM_SLEEP?

```
  0      DISABLE
  1      LIGHT
  2      MAC_IF                      (R)
```

ATH:00515Ch - RTC_AXI_AHB_BRIDGE ;hw6 only

```
  0-1    hw6: MAX_BEATS
```

ATH:005160h - UNIFIED_MAC_REVID (R) ;hw6 only

```
  0-31   hw6: VALUE                        (R)
```

DSi Atheros Wifi - Internal I/O - 0xx240h - RTC/Clock SYNC (hw6)

ATH:xxx240h - RTC_SYNC_RESET

```
  0      RESET_L
```

ATH:xxx244h - RTC_SYNC_STATUS

```
  0      SHUTDOWN_STATE              (R)
  1      ON_STATE                    (R)
  2      SLEEP_STATE                 (R)
  3      WAKEUP_STATE                (R)
  4      WRESET                      (R)
  5      PLL_CHANGING                (R)
```

ATH:xxx248h - RTC_SYNC_DERIVED

```
  0      BYPASS
  1      FORCE
  2      FORCE_SWREG_PWD             (W)
  3      FORCE_LPO_PWD               (W)
```

ATH:xxx24Ch - RTC_SYNC_FORCE_WAKE

```
  0      ENABLE                      (R)
  1      INTR
```

ATH:xxx250h - RTC_SYNC_INTR_CAUSE

```
  0      SHUTDOWN_STATE
  1      ON_STATE
  2      SLEEP_STATE
  3      WAKEUP_STATE
  4      SLEEP_ACCESS
  5      PLL_CHANGING
```

ATH:xxx254h - RTC_SYNC_INTR_ENABLE

0 SHUTDOWN_STATE
1 ON_STATE
2 SLEEP_STATE
3 WAKEUP_STATE
4 SLEEP_ACCESS
5 PLL_CHANGING

ATH:xxx258h - RTC_SYNC_INTR_MASK
0 SHUTDOWN_STATE
1 ON_STATE
2 SLEEP_STATE
3 WAKEUP_STATE
4 SLEEP_ACCESS
5 PLL_CHANGING

DSi Atheros Wifi - Internal I/O - 006000h - WLAN Coex (MCI) (hw6)

ATH:006000h - MCI_COMMAND0 ;hw6
0-7 HEADER
8-12 LEN
13 DISABLE_TIMESTAMP

ATH:006004h - MCI_COMMAND1 ;hw6
0-31 ADDR

ATH:006008h - MCI_COMMAND2 ;hw6
0 RESET_TX
1 RESET_RX
2-9 RESET_RX_NUM_CYCLES

ATH:00600Ch - MCI_RX_CTRL ;hw6
0 DISABLE_TIMESTAMP
1 DISABLE_MAXGAIN_RESET
2 DISABLE_MAXGAIN_WBTIMER_RESET

ATH:006010h - MCI_TX_CTRL ;hw6
0-1 CLK_DIV
2 DISABLE_LNA_UPDATES
3-23 GAIN_UPDATE_FREQ
24-27 GAIN_UPDATE_NUM

ATH:006014h - MCI_MSG_ATTRIBUTES_TABLE ;hw6
0-15 CHECKSUM_EN
16-31 INVALID_HDR

ATH:006018h - MCI_SCHD_TABLE_0 ;hw6
0-31 BASE_ADDR

ATH:00601Ch - MCI_SCHD_TABLE_1 ;hw6
0-15 OWN
16-31 SW_REQ_OWN

ATH:006020h - MCI_GPM_0 ;hw6
0-31 START_ADDR

ATH:006024h - MCI_GPM_1 ;hw6
0-15 LEN

```
16-31  WRITE_PTR                    (R)
```

ATH:006028h - MCI_INTERRUPT_RAW ;Interrupt Flags ;hw6
ATH:00602Ch - MCI_INTERRUPT_EN ;Interrupt Enable ;hw6

```
0      SW_MSG_DONE
1      CPU_INT_MSG
2      RX_CKSUM_FAIL
3      RX_INVALID_HDR
4      RX_HW_MSG_FAIL
5      RX_SW_MSG_FAIL
7      TX_HW_MSG_FAIL
8      TX_SW_MSG_FAIL
9      RX_MSG
10     REMOTE_SLEEP_UPDATE
11-26  BT_PRI
27     BT_PRI_THRESH
28     BT_FREQ
29     BT_STOMP
```

ATH:006030h - MCI_REMOTE_CPU_INT ;Flags ;hw6
ATH:006034h - MCI_REMOTE_CPU_INT_EN ;Enable ;hw6

```
0-31   BODY
```

ATH:006038h - MCI_INTERRUPT_RX_MSG_RAW ;Flags ;hw6
ATH:00603Ch - MCI_INTERRUPT_RX_MSG_EN ;Enable ;hw6

```
0      REMOTE_RESET
1      LNA_CTRL
2      CONT_NACK
3      CONT_INFO
4      CONT_RST
5      SCHD_INFO
6      CPU_INT
8      GPM
9      LNA_INFO
10     SYS_SLEEPING
11     SYS_WAKING
12     REQ_WAKE
```

ATH:006040h - MCI_CPU_INT ;hw6

```
0-31   MSG
```

ATH:006044h - MCI_RX_STATUS ;hw6

```
8-11   SCHD_MSG_INDEX              (R)
12     REMOTE_SLEEP                (R)
```

ATH:006048h - MCI_CONT_STATUS ;hw6

```
0-7    RSSI_POWER                  (R)
8-15   PRIORITY                    (R)
16     TX                          (R)
17-20  LINKID                      (R)
21-27  CHANNEL                     (R)
28-31  OWNER                       (R)
```

ATH:00604Ch - MCI_BT_PRI0 ;hw6
ATH:006050h - MCI_BT_PRI1 ;hw6
ATH:006054h - MCI_BT_PRI2 ;hw6
ATH:006058h - MCI_BT_PRI3 ;hw6

```
0-7    VAL0
8-15   VAL1
16-23  VAL2
24-31  VAL3
```

ATH:00605Ch - MCI_BT_PRI ;hw6
```
0-7     THRESH
```

ATH:006060h - MCI_WL_FREQ0 ;hw6
```
0-31    MASK
```

ATH:006064h - MCI_WL_FREQ1 ;hw6
```
0-31    MASK
```

ATH:006068h - MCI_WL_FREQ2 ;hw6
```
0-15    MASK
```

ATH:00606Ch - MCI_GAIN ;hw6
```
0-7     OFFSET1
8-15    OFFSET2
```

ATH:006070h - MCI_WBTIMER1 ;hw6
ATH:006074h - MCI_WBTIMER2 ;hw6
ATH:006078h - MCI_WBTIMER3 ;hw6
ATH:00607Ch - MCI_WBTIMER4 ;hw6
```
0-31    TARGET
```

ATH:006080h - MCI_MAXGAIN ;hw6
```
0-7     GAIN1
8-15    GAIN2
16-23   GAIN3
24-31   GAIN4
```

ATH:0060ACh - BTCOEX_CTRL ;hw6
```
2       MCI_MODE_ENABLE
```

ATH:00614Ch - BTCOEX_CTRL2 ;hw6
```
0-7     RESERVED2
8-10    OBS_SEL
```

ATH:006254h - BTCOEX_DBG ;hw6
```
0-31    OBS                             (R)
```

ATH:006258h - MCI_LAST_HW_MSG_HDR ;hw6
```
0-7     HDR                             (R)
8-10    LEN                             (R)
```

ATH:00625Ch - MCI_LAST_HW_MSG_BDY ;hw6
```
0-31    BDY                             (R)
```

ATH:006260h - MCI_MAXGAIN_RST ;hw6
```
0-31    TARGET
```

DSi Atheros Wifi - Internal I/O - 00x000h - Bluetooth Coex (hw4/hw6)

ATH:004218h/007000h - BTCOEXCTRL - hw4/hw6
```
0-7     GAP
8       CLK_CNT_EN
9       FRAME_CNT_EN
```

```
  10     IDLE_CNT_EN
  11     SYNC_DET_EN
  12-17  PRIORITY_TIME
  18-22  FREQ_TIME
  23-24  PTA_MODE
  25     WBSYNC_ON_BEACON
  26     hw4: WBTIMER_ENABLE            ;hw4 only
  27     unspecified
  28     hw6: RFGAIN_VALID_SRC          ;hw6 only
```

ATH:004228h/007010h - BTCOEX0 ;SYNC_DUR - hw4/hw6

```
  0-7    SYNC_DUR
```

ATH:00422Ch/007014h - BTCOEX1 ;CLK_THRES - hw4/hw6

```
  0-20   CLK_THRES
```

ATH:004230h/007018h - BTCOEX2 ;FRAME_THRES - hw4/hw6

```
  0-7    FRAME_THRES
```

ATH:004234h/00701Ch - BTCOEX3 ;CLK_CNT - hw4/hw6

```
  0-20   CLK_CNT
```

ATH:004238h/007020h - BTCOEX4 ;FRAME_CNT - hw4/hw6

```
  0-7    FRAME_CNT
```

ATH:00423Ch/007024h - BTCOEX5 ;IDLE_CNT - hw4/hw6

```
  0-15   IDLE_CNT
```

ATH:004240h/007028h - BTCOEX6 ;IDLE_RESET_LVL_BITMAP - hw4/hw6

```
  0-31   IDLE_RESET_LVL_BITMAP
```

ATH:004244h/00702Ch - LOCK - hw4/hw6

```
  0-7    TUNLOCK_MASTER
  8-15   TLOCK_MASTER
  16-23  TUNLOCK_SLAVE
  24-31  TLOCK_SLAVE
```

ATH:00421Ch/007004h - WBSYNC_PRIORITY1 - hw4/hw6
ATH:004220h/007008h - WBSYNC_PRIORITY2 - hw4/hw6
ATH:004224h/00700Ch - WBSYNC_PRIORITY3 - hw4/hw6
ATH:004248h/007030h - NOLOCK_PRIORITY - hw4/hw6

```
  0-31   BITMAP
```

ATH:00424Ch/007034h - WBSYNC - hw4/hw6
ATH:004250h/007038h - WBSYNC1 - hw4/hw6
ATH:004254h/00703Ch - WBSYNC2 - hw4/hw6
ATH:004258h/007040h - WBSYNC3 - hw4/hw6

```
  0-31   BTCLOCK                        (R) (read-only, according to hw6)
```

ATH:00425Ch/007044h - WB_TIMER_TARGET - hw4/hw6

```
  0-31   VALUE
```

ATH:004260h/007048h - WB_TIMER_SLOP - hw4/hw6

```
  0-9    VALUE
```

ATH:004264h/00704Ch - BTCOEX_INT_EN - hw4/hw6
ATH:004268h/007050h - BTCOEX_INT_STAT - hw4/hw6

```
  0      CLK_CNT
```

```
1       FRAME_CNT
2       END
3       SYNC
4       NOSYNC
5       BTPRIORITY       ;<-- for INT_STAT (but, N/A for INT_EN)       (R)
6       BTPRIORITY_STOMP ;<-- for INT_STAT (but, N/A for INT_EN)       (R)
7       WB_TIMER
8       I2C_MESG_RECV    ;<-- for INT_STAT  (but, "ST_MESG_RECV" for INT_EN?)
9       I2C_MESG_SENT
10      I2C_TX_FAILED
11      I2C_RECV_OVERFLOW
```

ATH:00426Ch/007054h - BTPRIORITY_INT_EN - hw4/hw6
ATH:004270h/007058h - BTPRIORITY_INT_STAT - hw4/hw6

```
0-31    BITMAP
```

ATH:004274h/00705Ch - BTPRIORITY_STOMP_INT_EN - hw4/hw6
ATH:004278h/007060h - BTPRIORITY_STOMP_INT_STAT - hw4/hw6

```
0-31    BITMAP
```

ATH:004294h/007064h - ST_64_BIT - hw4/hw6

```
0       MODE
1-5     SOC_CLK_DIVIDE_RATIO
6       CLOCK_GATE
7       DRIVE_MODE
8       REQ_ACK_NOT_PULLED_DOWN
9-26    TIMEOUT
```

ATH:004298h/007068h - MESSAGE_WR - hw4/hw6
ATH:0042A0h/007070h - MESSAGE_RD - hw4/hw6

```
0-31    TYPE
```

ATH:00429Ch/00706Ch - MESSAGE_WR_P - hw4/hw6
ATH:0042A4h/007074h - MESSAGE_RD_P - hw4/hw6

```
0-31    PARAMETER

_________ below hw6 only _________
```

ATH:007078h - BTPRIORITY_INT - hw6 only

```
0-7     DELAY
```

ATH:00707Ch - SCO_PARAMS - hw6 only

```
0-13    PERIOD
14-23   SLOP
```

ATH:007080h - SCO_PRIORITY - hw6 only

```
0-31    BITMAP
```

ATH:007084h - SCO_SYNC - hw6 only

```
0-31    BTCLOCK
```

ATH:007088h - BTCOEX_RAW_STAT - hw6 only

```
0       CLK_CNT
1       FRAME_CNT
2       END
3       SYNC
4       NOSYNC
7       WB_TIMER
```

ATH:00708Ch - BTPRIORITY_RAW_STAT - hw6 only
```
  0-31   BITMAP
```

ATH:007090h - BTPRIORITY_STOMP_RAW_STAT - hw6 only
```
  0-31   BITMAP
```

DSi Atheros Wifi - Internal I/O - 00x000h - Memory Control (hw2/hw4/hw6)

TCAM/BCAM (Ternary/Binary Content Addressable Memory) (ROM Patches)
TCAM/BCAM registers are allowing to patch ROM (in a similar fashion as Game Genie cheat devices). The ROM patches can be initialized via BMI commands:
DSi Atheros Wifi - BMI Bootloader Commands
Many ROM functions are called via a Table in RAM, which can be patched without needing the TCAM/BCAM feature (actually, the DSi's AR6002 firmware is patching only three of that RAM Table entries, and leaves the TCAM feature completely unused).

```
_____________________ hw2 ROM Patches (TCAM) _____________________
```

ATH:008000h..807Ch - (WLAN_)MC_TCAM_VALID[0..31] ;hw2
```
  0      BIT    (?=Patch Enable)
```

ATH:008080h..80FCh - (WLAN_)MC_TCAM_MASK[0..31] ;hw2
```
  0-2    SIZE   (... patch area, selectable 32-bytes or bigger or so?)
```
The eight size settings are probably 20h,40h,80h,100h,200h,400h,800h,1000h.

ATH:008100h..817Ch - (WLAN_)MC_TCAM_COMPARE[0..31] ;hw2
```
  5-21   KEY    (Patch ROM Address in 32-byte steps) (probably 0E0000h and up?)
```

ATH:008180h..81FCh - (WLAN_)MC_TCAM_TARGET[0..31] ;hw2
```
  5-21   ADDR   (Patch RAM Address in 32-byte steps) (probably 100000h and up?)
```

```
_____________________ hw4 ROM Patches (BCAM) _____________________
```

The hw4 ROM patching is done in 32bit Data units, but 16bit/24bit Xtensa opcodes aren't neccessarily 32bit aligned, so one may need 1-2 patch slots per opcode.

ATH:008000h..0081FCh - WLAN_MC_BCAM_VALID[0..127] ;hw4
```
  0      BIT    some "bit" (128 x 1bit)    (?=Patch Enable)
  1-31   -
```

ATH:008200h..0083FCh - WLAN_MC_BCAM_COMPARE[0..127] ;hw4
```
  0-1    -
  2-19   KEY    some "key" (128 x 18bit)   (Patch Address in 4-byte steps)
  20-31  -
```

ATH:008400h..0085FCh - WLAN_MC_BCAM_TARGET[0..127] ;hw4
```
  0-31   INST   some "inst" (128 x 32bit)  (Patch Data)
```

ATH:008610h - WLAN_BCAM_CONFLICT_ERROR ;hw4
```
  0      DPORT_FLAG
  1      IPORT_FLAG
  2-31   -
```

hw6 ROM Patches (?)

Unknown if or how ROM Patches are supported on hw6 (the hw6 source code doesn't define any TCAM/BCAM registers).

ADDR_ERROR Registers

ATH:008200h - (WLAN_)ADDR_ERROR_CONTROL ;hw2
ATH:008600h - WLAN_APB_ADDR_ERROR_CONTROL ;hw4
ATH:010018h - WLAN_APB_ADDR_ERROR_CONTROL ;hw6

0	ENABLE
1	QUAL_ENABLE
2-31	-

ATH:008204h - (WLAN_)ADDR_ERROR_STATUS ;hw2
ATH:008604h - WLAN_APB_ADDR_ERROR_STATUS ;hw4
ATH:01001Ch - WLAN_APB_ADDR_ERROR_STATUS ;hw6

0-24	ADDRESS
25	WRITE
26-31	-

ATH:008608h - WLAN_AHB_ADDR_ERROR_CONTROL ;hw4
ATH:010020h - WLAN_AHB_ADDR_ERROR_CONTROL ;hw6

0	ENABLE
1-31	-

ATH:00860Ch - WLAN_AHB_ADDR_ERROR_STATUS ;hw4
ATH:010024h - WLAN_AHB_ADDR_ERROR_STATUS ;hw6

0-23	ADDRESS
24-29	-
30	MBOX
31	MAC

hw4 MISC Registers

ATH:008614h - WLAN_CPU_PERF_CNT ;hw4

0	EN
1-31	-

ATH:008618h - WLAN_CPU_INST_FETCH ;hw4
ATH:00861Ch - WLAN_CPU_DATA_FETCH ;hw4

0-31	CNT

ATH:008620h - WLAN_CPU_RAM1_CONFLICT ;hw4
ATH:008624h - WLAN_CPU_RAM2_CONFLICT ;hw4
ATH:008628h - WLAN_CPU_RAM3_CONFLICT ;hw4
ATH:00862Ch - WLAN_CPU_RAM4_CONFLICT ;hw4

0-11	CNT
12-31	-

hw6 MISC Registers

ATH:010028h - WLAN_AHB_CONFIG ;hw6

0	MAX_BURST_4
1	MAX_BURST_8
2	MAX_BURST_16

ATH:01002Ch - WLAN_MEMORY_MAP ;hw6

```
0      ONE_IRAM_BANK
1      TWO_IRAM_BANKS
2      THREE_IRAM_BANKS
3      FOUR_IRAM_BANKS

_____________________ Xtensa CPU _____________________
```

Xtensa Region/MMU (hw2)
The Xtensa CPU is additionally having a Region/MMU unit with ITLB and DTLB. The AR6002 ROM is doing some basic initialization via mmu-opcodes:

```
set ITLB[(0..7)*20000000h] to values (1,2,2,2,2,2,2,2)
set DTLB[(0..7)*20000000h] to values (1,2,2,2,2,2,2,2)
```

Alongsides, it's issuing an "isync" opcode after setting the first ITLB entry, and a "dsync" opcode after setting the last DTLB entry. After that initialization, the ROM and Firmware aren't using any further mmu-opcodes.

DSi Atheros Wifi - Internal I/O - 00C000h - Serial UART (hw2/hw4/hw6)

The UART_xxx registers are used to output TTY messages (ASCII strings) when enabled in the "Host Interest" area:

```
LOCAL_SCRATCH[0].bit1 AR6K_OPTION_SERIAL_ENABLE --> TTY master enable
targaddr[14h] hi_serial_enable --> enable additional TTY msg's during BMI
targaddr[60h] hi_desired_baud_rate --> for TTY/UART (default=9600 decimal)
targaddr[C4h] hi_console_flags - whatever, UART related, maybe newer firmware

_____________________ hw2 UART Registers _____________________
```

Texas Instruments TL16C550AN - Asynchronous Communications Element (ACE)
The hw2 UART is based on the TL16C550AN chip (which is also found in the SNES "Exertainment Bicycle" add-on).

ATH:00C000h (when DLAB=0) - (WLAN_UART_)RBR - RX Data FIFO (R) ;hw2
ATH:00C020h (when DLAB=0) - (WLAN_UART_)SRBR (mirror of RBR?) ;hw2

```
0-7  Data (with 16-byte FIFO)
```

ATH:00C000h (when DLAB=0) - (WLAN_UART_)THR - TX Data FIFO (W) ;hw2

```
0-7  Data (with 16-byte FIFO)
```

ATH:00C004h (when DLAB=0) - (WLAN_UART_)IER - Interrupt Control (R/W) ;hw2

```
0    ERBFI Received Data Available Interrupt           (0=Disable, 1=Enable)
1    ETBEI Transmitter Holding Register Empty Interrupt (0=Disable, 1=Enable)
2    ELSI  Receiver Line Status Interrupt              (0=Disable, 1=Enable)
3    EDDSI Modem Status Interrupt                      (0=Disable, 1=Enable)
4-7  -     Not used (always zero)
```

ATH:00C000h (when DLAB=1) - (WLAN_UART_)DLL - Baudrate Divisor LSB (R/W) ;hw2
ATH:00C004h (when DLAB=1) - (WLAN_UART_)DLH - Baudrate Divisor MSB (R/W) ;hw2

```
0-7  Divisor Latch LSB/MSB, should be set to "divisor = XIN / (baudrate*16)"
```

ATH:00C008h - (WLAN_UART_)IIR - Interrupt Status (R) ;hw2
ATH:00C028h - (WLAN_UART_)SIIR (mirror or IIR?) ;hw2

```
0    Interrupt Pending Flag (0=Pending, 1=None)                      ;\IID
1-3  Interrupt ID, 3bit     (0..7=see below) (always 00h when Bit0=1) ;/
4-5  Not used (always zero)
6    FIFOs Enabled (always zero in TL16C450 mode) ;\these bits have same
7    FIFOs Enabled (always zero in TL16C450 mode) ;/value as "FIFO Enable"
```

The 3bit Interrupt ID can have following values:

```
ID Prio Expl.
00h 4   Handshaking inputs CTS,DSR,RI,DCD have changed      (Ack: Read MSR)
01h 3   Transmitter Holding Register Empty     (Ack: Write THR or Read IIR)
02h 2   RX FIFO has reached selected trigger level          (Ack: Read RBR)
03h 1   RX Overrun/Parity/Framing Error, or Break Interrupt (Ack: Read LSR)
06h 2   RX FIFO non-empty & wasn't processed for longer time(Ack: Read RBRh)
```

Interrupt ID values 04h,05h,07h are not used.

ATH:00C008h - (WLAN_UART_)FCR - FIFO Control (W) ;hw2

```
0    FIFO Enable (0=Disable, 1=Enable) (Enables access to FIFO related bits)
1    Receiver FIFO Reset      (0=No Change, 1=Clear RX FIFO) (RCVR_FIFO_RST)
2    Transmitter FIFO Reset   (0=No Change, 1=Clear TX FIFO) (XMIT_FIFO_RST)
3    DMA Mode Select (Mode for /RXRDY and /TXRDY) (0=Mode 0, 1=Mode 1)
4-5  Not used (should be zero)
6-7  Receiver FIFO Trigger    (0..3 = 1,4,8,14 bytes) (RCVR_TRIG)
```

ATH:00C00Ch - (WLAN_UART_)LCR - Character Format Control (R/W) ;hw2

```
0-1  Character Word Length    (0..3 = 5,6,7,8 bits) (CLS)
2    Number of Stop Bits      (0=1bit, 1=2bit; for 5bit chars: only 1.5bit)
3    Parity Enable (PEN)      (0=None, 1=Enable Parity or 9th data bit)
4    Parity Type/9th Data bit (0=Odd, 1=Even) (EPS)
5    Unused in hw2? ;for TL16C550AN: Bit4-5 can be 2=Set9thBit, 3=Clear9thBit
6    Set Break                (0=Normal, 1=Break, Force SOUT to Low)
7    Divisor Latch Access     (0=Normal I/O, 1=Divisor Latch I/O) (DLAB)
```

ATH:00C010h - (WLAN_UART_)MCR - Handshaking Control (R/W) ;hw2

```
0    DTR  Output Level for /DTR pin  (Data Terminal Ready) (0=High, 1=Low)
1    RTS  Output Level for /RTS pin  (Request to Send)     (0=High, 1=Low)
2    OUT1 Output Level for /OUT1 pin (General Purpose)     (0=High, 1=Low)
3    OUT2 Output Level for /OUT2 pin (General Purpose)     (0=High, 1=Low)
4/5? LOOP Loopback Mode (0=Normal, 1=Testmode, loopback TX to RX)
5-7  Not used (always zero)
```

The Loopback bit should be Bit4 (according to TL16C550AN datasheet), but hw2 source code claims it to be in bit5.

ATH:00C014h - (WLAN_UART_)LSR - RX/TX Status (R) (W=don't do) ;hw2
ATH:00C034h - (WLAN_UART_)SLSR (mirror of LSR?) ;hw2

```
0    RX Data Ready (DR)       (0=RX FIFO Empty, 1=RX Data Available)
1    RX Overrun Error (OE)    (0=Okay, 1=Error) (RX when RX FIFO Full)
2    RX Parity Error (PE)     (0=Okay, 1=Error) (RX parity bad)
3    RX Framing Error (FE)    (0=Okay, 1=Error) (RX stop bit bad)
4    RX Break Interrupt (BI)  (0=Normal, 1=Break) (RX line LOW for long time)
5    Transmitter Holding Register (THRE) (1=TX FIFO is empty)
6    Transmitter Empty (TEMT) (0=No, 1=Yes, TX FIFO and TX Shift both empty)
7    At least one Overrun/Parity/Framing Error in RX FIFO (0=No, 1=Yes/Error)
```

Bit7 is always zero in TL16C450 mode. Bit1-3 are automatically cleared after reading. In FIFO mode, bit2-3 reflect to status of the current (=oldest) character in the FIFO (unknown/unclear if bit2-3 are also auto-cleared when in FIFO mode).
Note: The AR6002 BIOS ROM is using "SLSR" (instead of "LSR") for testing bit0,5,6. And, before each read from SLSR register, the AR6002 BIOS BIOS does first write 0 to SLSR (for whatever unknown purpose). Basically, "SLSR" (and other "Sxxx" registers) seems to be some sort of a mirror of "LSR" (and other "xxx" registers)? Maybe one of them omits automatic IRQ acknowledge or so?

ATH:00C018h - (WLAN_UART_)MSR - Handshaking Status (R) (W=don't do) ;hw2
ATH:00C038h - (WLAN_UART_)SMSR (mirror or MSR?) ;hw2

```
0    DCTS Change flag for /CTS pin ;ClearToSend       ;\change flags (0=none,
1    DDSR Change flag for /DSR pin ;DataSetReady      ; 1=changed since last
2    TERI Change flag for /RI pin  ;RingIndicator     ; read) (automatically
3    DDCD Change flag for /DCD pin ;DataCarrierDetect ;/cleared after read)
```

```
  4    CTS  Input Level on /CTS pin ;ClearToSend        ;\
  5    DSR  Input Level on /DSR pin ;DataSetReady       ; current levels
  6    RI   Input Level on /RI pin  ;RingIndicator      ; (inverted ?)
  7    DCD  Input Level on /DCD pin ;DataCarrierDetect  ;/
```

ATH:00C01Ch - (WLAN_UART_)SCR - Scratch (R/W) ;hw2

```
  0-7  General Purpose Storage (eg. read/write-able for UART chip detection)
```

ATH:00C02Ch - (WLAN_UART_?)MWR ;whatever "M Write Register?" ;hw2
ATH:00C03Ch - (WLAN_UART_?)MRR ;whatever "M Read Register?" ;hw2

```
  0-31 whatever... 32bit wide (unlike other UART registers) (?) (UART related?)
```

_____ hw4/hw6 UART Registers _____

Multiple UARTs ;hw4/hw6

There appear to be multiple hw4/hw6 UARTs: one normal, one for debug, one for hw6:

```
  WLAN_UART_BASE_ADDRESS     = 0000C000h ;hw4/hw6
  WLAN_DBG_UART_BASE_ADDRESS = 0000D000h ;hw4/hw6
  WLAN_UART2_BASE_ADDRESS    = 00054C00h ;hw6
```

Maybe the UARTs are all using the same register format with differerent base?

ATH:00C000h - UART_DATA ;hw4/hw6

```
  0-7    TXRX_DATA
  8      RX_CSR
  9      TX_CSR
```

ATH:00C004h - UART_CONTROL ;hw4/hw6

```
  0      PARITY_EVEN
  1      PARITY_ENABLE
  2      IFC_DCE
  3      IFC_ENABLE
  4      FLOW_INVERT
  5      FLOW_ENABLE
  6      DMA_ENABLE
  7      RX_READY_ORIDE
  8      TX_READY_ORIDE
  9      SERIAL_TX_READY
  10     RX_BREAK
  11     TX_BREAK
  12     HOST_INT
  13     HOST_INT_ENABLE
  14     TX_BUSY
  15     RX_BUSY
```

ATH:00C008h - UART_CLKDIV ;hw4/hw6

```
  0-15   CLK_STEP
  16-23  CLK_SCALE
```

ATH:00C00Ch - UART_INT ;hw4/hw6
ATH:00C010h - UART_INT_EN ;hw4/hw6

```
  0      RX_VALID_INT
  1      TX_READY_INT
  2      RX_FRAMING_ERR_INT
  3      RX_OFLOW_ERR_INT
  4      TX_OFLOW_ERR_INT
  5      RX_PARITY_ERR_INT
  6      RX_BREAK_ON_INT
  7      RX_BREAK_OFF_INT
  8      RX_FULL_INT
  9      TX_EMPTY_INT
```

DSi Atheros Wifi - Internal I/O - 00E000h - UMBOX Registers (hw4/hw6)

ATH:00E000h..00E004h - UMBOX_FIFO[0..1]
```
0-8    DATA   ... uh, twice[0..1], with 9bit each ?
```

ATH:00E008h - UMBOX_FIFO_STATUS
```
0      RX_FULL
1      RX_EMPTY
2      TX_FULL
3      TX_EMPTY
```

ATH:00E00Ch - UMBOX_DMA_POLICY
```
0      RX_ORDER
1      RX_QUANTUM
2      TX_ORDER
3      TX_QUANTUM
```

ATH:00E010h - UMBOX0_DMA_RX_DESCRIPTOR_BASE
ATH:00E018h - UMBOX0_DMA_TX_DESCRIPTOR_BASE
```
2-27   ADDRESS
```

ATH:00E014h - UMBOX0_DMA_RX_CONTROL
ATH:00E01Ch - UMBOX0_DMA_TX_CONTROL
```
0      STOP
1      START
2      RESUME
```

ATH:00E020h - UMBOX_FIFO_TIMEOUT
```
0-7    VALUE
8      ENABLE_SET
```

ATH:00E024h - UMBOX_INT_STATUS
ATH:00E028h - UMBOX_INT_ENABLE
```
0      RX_NOT_FULL
1      TX_NOT_EMPTY
2      RX_UNDERFLOW
3      TX_OVERFLOW
4      HCI_SYNC_ERROR
5      TX_DMA_COMPLETE
6      TX_DMA_EOM_COMPLETE
7      RX_DMA_COMPLETE
8      HCI_FRAMER_OVERFLOW
9      HCI_FRAMER_UNDERFLOW
```

ATH:00E02Ch - UMBOX_DEBUG
```
0-2    SEL
```

ATH:00E030h - UMBOX_FIFO_RESET
```
0      INIT
```

ATH:00E034h - UMBOX_HCI_FRAMER
```
0-1    CONFIG_MODE
2      OVERFLOW
3      UNDERFLOW
4      SYNC_ERROR
```

```
5       ENABLE
6       CRC_OVERRIDE
```

DSi Atheros Wifi - Internal I/O - 010000h - Serial I2C/SPI (hw2/hw4/hw6)

These registers are providing a general purpose I2C/SPI serial bus. The SI_xxx registers are exactly same in hw2/hw4/hw6.
In the DSi, they are used in I2C mode - for reading wifi calibration data:
DSi Atheros Wifi I2C EEPROM

ATH:010000h - SI_CONFIG

```
0-3     DIVIDER             (probably transfer rate, should be 6 on DSi)
4       INACTIVE_CLK        (whatever, should be 1 for I2C)
5       INACTIVE_DATA       (whatever, should be 1 for I2C)
6       POS_DRIVE           (whatever, should be zero for I2C)
7       POS_SAMPLE          (whatever, should be 1 for I2C)
8-15    -
16      I2C                 (0=SPI, 1=I2C)
17      -
18      BIDIR_OD_DATA       (whatever, should be 1 for I2C)
19      ERR_INT             (whatever, enable or status?)
```

On DSi, this is register is set to 500B6h.

ATH:010004h - SI_CS

```
0-3     TX_CNT     Number of TX bytes (0..8) (should be 1..8 for I2C device)
4-7     RX_CNT     Number of RX bytes (0..8)
8       START      Write 1 to start transfer
9       DONE_INT   Status (0=Busy, 1=Done/Okay)
10      DONE_ERR   Status (1=Error)
11-13   BIT_CNT_IN_LAST_BYTE (0=Normal/8bit, 1..7=whatever)
```

For I2C with TX_CNT and RX_CNT both nonzero: TX data is transferred first.
Unknown when the DONE flags are cleared (possibly when writing 0 to bit9,10, or when writing 1 to bit8, or maybe automatically after reading).

ATH:010008h..01000Ch - SI_TX_DATA0/SI_TX_DATA1

```
0-7     DATA0   1st TX byte (device number in case of I2C mode)
8-15    DATA1   2nd TX byte (if any)
16-23   DATA2   ...
24-31   DATA3   ..
32-39   DATA4
40-47   DATA5
48-55   DATA6
56-63   DATA7
```

ATH:010010h..010014h - SI_RX_DATA0/SI_RX_DATA1

```
0-7     DATA0   1st RX byte (if any)
8-15    DATA1   2nd RX byte (if any)
16-23   DATA2   ...
24-31   DATA3   ..
32-39   DATA4
40-47   DATA5
48-55   DATA6
56-63   DATA7
```

DSi Atheros Wifi - Internal I/O - 014000h - GPIO 18/26/57 pin

(hw2/hw4/hw6)

ATH:014000h/014000h/014000h - WLAN_GPIO_OUT ;GPIO Data Out ;hw2/hw4/hw6
ATH:014004h/014004h/014004h - WLAN_GPIO_OUT_W1TS ;Write-1-to-Set ;hw2/hw4/hw6
ATH:014008h/014008h/014008h - WLAN_GPIO_OUT_W1TC ;Write-1-to-Clr ;hw2/hw4/hw6
ATH:01400Ch - WLAN_GPIO_OUT_HIGH ;for pin32 and up ;hw6
ATH:014010h - WLAN_GPIO_OUT_W1TS_HIGH ;for pin32 and up ;hw6
ATH:014014h - WLAN_GPIO_OUT_W1TC_HIGH ;for pin32 and up ;hw6

```
  0-17   hw2: DATA (for pin 0..17)
  0-25   hw4: DATA (for pin 0..25)
  0-63   hw6: DATA (for pin 0..56) (and bit57-63=unused or so?)
```

ATH:01400Ch/01400Ch/014018h - WLAN_GPIO_ENABLE ;GPIO Out Enable ;hw2/hw4/hw6
ATH:014010h/014010h/01401Ch - WLAN_GPIO_ENABLE_W1TS ;Wr-1-to-Set ;hw2/hw4/hw6
ATH:014014h/014014h/014020h - WLAN_GPIO_ENABLE_W1TC ;Wr-1-to-Clr ;hw2/hw4/hw6
ATH:014024h - WLAN_GPIO_ENABLE_HIGH ;for pin32 and up ;hw6
ATH:014028h - WLAN_GPIO_ENABLE_W1TS_HIGH ;for pin32 and up ;hw6
ATH:01402Ch - WLAN_GPIO_ENABLE_W1TC_HIGH ;for pin32 and up ;hw6

```
  0-17   hw2: DATA (for pin 0..17)
  0-25   hw4: DATA (for pin 0..25)
  0-63   hw6: DATA (for pin 0..56) (and bit57-63=unused or so?)
```

ATH:014018h/014018h/014030h - WLAN_GPIO_IN - GPIO Data In ;hw2/hw4/hw6
ATH:014038h - WLAN_GPIO_IN_HIGH ;for pin32 and up ;hw6

```
  0-17   hw2: DATA (for pin 0..17)
  0-25   hw4: DATA (for pin 0..25)
  0-63   hw6: DATA (for pin 0..56) (and bit57-63=unused or so?)
```

ATH:01401Ch/01401Ch/014034h - WLAN_GPIO_STATUS - GPIO Interrupt ;hw2/hw4/hw6
ATH:014020h/014020h/014040h - WLAN_GPIO_STATUS_W1TS ;Wr-1-to-Set ;hw2/hw4/hw6
ATH:014024h/014024h/014044h - WLAN_GPIO_STATUS_W1TC ;Wr-1-to-Clr ;hw2/hw4/hw6
ATH:01403Ch - WLAN_GPIO_STATUS_HIGH ;for pin32 and up ;hw6
ATH:014048h - WLAN_GPIO_STATUS_W1TS_HIGH ;for pin32 and up ;hw6
ATH:01404Ch - WLAN_GPIO_STATUS_W1TC_HIGH ;for pin32 and up ;hw6

```
  0-17   hw2: INTERRUPT (for pin 0..17)
  0-25   hw4: INTERRUPT (for pin 0..25)
  0-63   hw6: INTERRUPT (for pin 0..56) (and bit57-63=unused or so?)
```

```
 _________________________________ hw2 GPIO ports _________________________________
```

ATH:014028h - GPIO_PIN0 ;GPIO0 Bluetooth coex BT_PRIORITY
ATH:01402Ch - GPIO_PIN1 ;GPIO1 Bluetooth coex WLAN_ACTIVE
ATH:014030h - GPIO_PIN2 ;GPIO2 Bluetooth coex BT_FREQUENCY ;I2C SCL
ATH:014034h - GPIO_PIN3 ;GPIO3 Bluetooth coex BT_ACTIVE ;I2C_SDA
ATH:014038h - GPIO_PIN4 ;GPIO4 SDIO/GSPI interface select
ATH:01403Ch - GPIO_PIN5 ;GPIO5 SDIO/GSPI interface select
ATH:014040h - GPIO_PIN6 ;GPIO6 -
ATH:014044h - GPIO_PIN7 ;GPIO7 TRST for JTAG debug
ATH:014048h - GPIO_PIN8 ;GPIO8 external 32kHz clock in
ATH:01404Ch - GPIO_PIN9 ;GPIO9 I2C SCL or SPI CLK
ATH:014050h - GPIO_PIN10 ;GPIO10 I2C SDA or SPI MISO
ATH:014054h - GPIO_PIN11 ;GPIO11 UART RXD or SPI MOSI
ATH:014058h - GPIO_PIN12 ;GPIO12 UART TXD or SPI /CS
ATH:01405Ch - GPIO_PIN13 ;GPIO13 Reset in for JTAG debug
ATH:014060h - GPIO_PIN14 ;GPIO14 UART CTS
ATH:014064h - GPIO_PIN15 ;GPIO15 UART RTS

ATH:014068h - GPIO_PIN16 ;GPIO16 -
ATH:01406Ch - GPIO_PIN17 ;GPIO17 -

```
0      SOURCE
1      -
2      PAD_DRIVER
3-6    -
7-9    INT_TYPE
10     WAKEUP_ENABLE
11-12  CONFIG
```

Note: The I2C EEPROM in DSi/3DS is accessed via Pin9/10 on AR6002, and Pin2/3 on AR6013/AR6014.

_____ hw4 GPIO ports _____

ATH:014028h - WLAN_GPIO_PIN0 ;GPIO0 Bluetooth coex BT_FREQUENCY
ATH:01402Ch - WLAN_GPIO_PIN1 ;GPIO1 Bluetooth coex WLAN_ACTIVE
ATH:014030h - WLAN_GPIO_PIN2 ;GPIO2 Bluetooth coex BT_ACTIVE
ATH:014034h - WLAN_GPIO_PIN3 ;GPIO3 Bluetooth coex BT_PRIORITY
ATH:014038h - WLAN_GPIO_PIN4 ;GPIO4 -
ATH:01403Ch - WLAN_GPIO_PIN5 ;GPIO5 JTAG TMS input
ATH:014040h - WLAN_GPIO_PIN6 ;GPIO6 JTAG TCK input
ATH:014044h - WLAN_GPIO_PIN7 ;GPIO7 JTAG TDI input
ATH:014048h - WLAN_GPIO_PIN8 ;GPIO8 JTAG TDO output
ATH:01404Ch - WLAN_GPIO_PIN9 ;GPIO9 SDIO CMD
ATH:014050h - WLAN_GPIO_PIN10 ;GPIO10 SDIO D3
ATH:014054h - WLAN_GPIO_PIN11 ;GPIO11 SDIO D2
ATH:014058h - WLAN_GPIO_PIN12 ;GPIO12 SDIO D1
ATH:01405Ch - WLAN_GPIO_PIN13 ;GPIO13 SDIO D0
ATH:014060h - WLAN_GPIO_PIN14 ;GPIO14 SDIO CLK
ATH:014064h - WLAN_GPIO_PIN15 ;GPIO15 HCI UART TXD
ATH:014068h - WLAN_GPIO_PIN16 ;GPIO16 HCI UART RTS
ATH:01406Ch - WLAN_GPIO_PIN17 ;GPIO17 HCI UART RXD
ATH:014070h - WLAN_GPIO_PIN18 ;GPIO18 HCI UART CTS
ATH:014074h - WLAN_GPIO_PIN19 ;GPIO19 SDIO/GSPI interface select
ATH:014078h - WLAN_GPIO_PIN20 ;GPIO20 SDIO/GSPI interface select
ATH:01407Ch - WLAN_GPIO_PIN21 ;GPIO21 external input sleep clock
ATH:014080h - WLAN_GPIO_PIN22 ;GPIO22 wake on wireless input (WOW)
ATH:014084h - WLAN_GPIO_PIN23 ;GPIO23 reference clk output to BT chip
ATH:014088h - WLAN_GPIO_PIN24 ;GPIO24 request clk from BT chip
ATH:01408Ch - WLAN_GPIO_PIN25 ;GPIO25 request reference clk (CLK_REQ)

```
0      SOURCE
1      -
2      PAD_DRIVER
3-4    PAD_STRENGTH   ;\pull/strength supported for PIN0..PIN22 only
5-6    PAD_PULL       ;/(bit3-6 are unused in PIN23..PIN25 registers)
7-9    INT_TYPE
10     WAKEUP_ENABLE
11-13  CONFIG
```

AR6003 datasheet assigns only the above stuff with a single UART, although the AR6003 should additionally support a DBG_UART (and I2C).

_____ hw6 GPIO ports _____

ATH:014050h - WLAN_GPIO_PIN0 ;GPIO0 or SDIO_CMD
ATH:014054h - WLAN_GPIO_PIN1 ;GPIO1 or SDIO_D3
ATH:014058h - WLAN_GPIO_PIN2 ;GPIO2 or SDIO_D2
ATH:01405Ch - WLAN_GPIO_PIN3 ;GPIO3 or SDIO_D1
ATH:014060h - WLAN_GPIO_PIN4 ;GPIO4 or SDIO_D0

ATH:014064h - WLAN_GPIO_PIN5 ;GPIO5 or SDIO_CLK
ATH:014068h - WLAN_GPIO_PIN6
ATH:01406Ch - WLAN_GPIO_PIN7
ATH:014070h - WLAN_GPIO_PIN8
ATH:014074h - WLAN_GPIO_PIN9
ATH:014078h - WLAN_GPIO_PIN10
ATH:01407Ch - WLAN_GPIO_PIN11
ATH:014080h - WLAN_GPIO_PIN12
ATH:014084h - WLAN_GPIO_PIN13
ATH:014088h - WLAN_GPIO_PIN14
ATH:01408Ch - WLAN_GPIO_PIN15
ATH:014090h - WLAN_GPIO_PIN16
ATH:014094h - WLAN_GPIO_PIN17
ATH:014098h - WLAN_GPIO_PIN18
ATH:01409Ch - WLAN_GPIO_PIN19
ATH:0140A0h - WLAN_GPIO_PIN20
ATH:0140A4h - WLAN_GPIO_PIN21
ATH:0140A8h - WLAN_GPIO_PIN22
ATH:0140ACh - WLAN_GPIO_PIN23
ATH:0140B0h - WLAN_GPIO_PIN24
ATH:0140B4h - WLAN_GPIO_PIN25
ATH:0140B8h - WLAN_GPIO_PIN26
ATH:0140BCh - WLAN_GPIO_PIN27
ATH:0140C0h - WLAN_GPIO_PIN28
ATH:0140C4h - WLAN_GPIO_PIN29
ATH:0140C8h - WLAN_GPIO_PIN30
ATH:0140CCh - WLAN_GPIO_PIN31
ATH:0140D0h - WLAN_GPIO_PIN32
ATH:0140D4h - WLAN_GPIO_PIN33
ATH:0140D8h - WLAN_GPIO_PIN34
ATH:0140DCh - WLAN_GPIO_PIN35
ATH:0140E0h - WLAN_GPIO_PIN36
ATH:0140E4h - WLAN_GPIO_PIN37
ATH:0140E8h - WLAN_GPIO_PIN38
ATH:0140ECh - WLAN_GPIO_PIN39
ATH:0140F0h - WLAN_GPIO_PIN40
ATH:0140F4h - WLAN_GPIO_PIN41
ATH:0140F8h - WLAN_GPIO_PIN42
ATH:0140FCh - WLAN_GPIO_PIN43
ATH:014100h - WLAN_GPIO_PIN44
ATH:014104h - WLAN_GPIO_PIN45
ATH:014108h - WLAN_GPIO_PIN46
ATH:01410Ch - WLAN_GPIO_PIN47
ATH:014110h - WLAN_GPIO_PIN48
ATH:014114h - WLAN_GPIO_PIN49
ATH:014118h - WLAN_GPIO_PIN50
ATH:01411Ch - WLAN_GPIO_PIN51
ATH:014120h - WLAN_GPIO_PIN52
ATH:014124h - WLAN_GPIO_PIN53
ATH:014128h - WLAN_GPIO_PIN54
ATH:01412Ch - WLAN_GPIO_PIN55
ATH:014130h - WLAN_GPIO_PIN56

```
  0       SOURCE
  1       -
```

```
  2      PAD_DRIVER
  3-4    PAD_STRENGTH
  5-6    PAD_PULL
  7-9    INT_TYPE
  10     WAKEUP_ENABLE
  11-14  CONFIG
```
AR6004 datasheet assigns only six SDIO signals to GPIO pins. However, signals similar as on AR6002/AR6003 should exist (I2C/SPI, UART, etc.), plus TWO additional UARTs).

_____ hw2/hw4/hw6 stuff _____

ATH:014078h/01409Ch/01413Ch - WLAN_SIGMA_DELTA ;hw2/hw4/hw6
```
  0-7    TARGET
  8-15   PRESCALAR      ;uh, scalar?
  16     ENABLE
```

ATH:01407Ch/0140A8h/01414Ch - WLAN_DEBUG_CONTROL ;hw2/hw4/hw6
```
  0      ENABLE                 ;-hw2/hw4/hw6
  1      hw2: OBS_OE_L          ;-hw2 only (bit1 removed in hw4/hw6)
```

ATH:014080h/0140ACh/014150h - WLAN_DEBUG_INPUT_SEL ;hw2/hw4/hw6
```
  0-3    SRC                    ;-hw2/hw4/hw6
  4-5    hw4/hw6: SHIFT         ;-hw4/hw6
```

ATH:014084h/0140B0h/014154h - WLAN_DEBUG_OUT ;hw2/hw4/hw6
```
  0-17   DATA (whatever) (always 18bit, no matter if GPIO with 18,25,57 pins)
```

ATH:0140F0h/0140B4h/014158h - WLAN_RESET_TUPLE_STATUS ;hw2/hw4/hw6
```
  0-7    PIN_RESET_TUPLE
  8-11   TEST_RESET_TUPLE
```

_____ hw4/hw6 stuff _____

ATH:014090h/014134h - SDIO ;hw4/hw6
ATH:014160h - SDIO2 ;hw6
ATH:014164h - SDHC ;hw6
```
  0      PINS_EN
```

ATH:014098h/014138h - WL_SOC_APB ;hw4/hw6
```
  0      TOGGLE
```

ATH:0140A0h/014140h - WL_BOOTSTRAP ;hw4/hw6
```
  0-22   hw4: STATUS (23bit) ;maybe for pin 0..22 (but not pin 23-25 ?)
  0-11   hw6: STATUS (12bit) ;maybe for pin 0..57 (with below "CORE_BOOTSTRAP")
  12     hw6: CPU_MBIST_EN
```

ATH:014144h - CORE_BOOTSTRAP_LOW ;hw6
```
  0-31   hw6: STATUS (32bit) (extra bits, expanding STATUS in "WL_BOOTSTRAP"?)
```

ATH:014148h - CORE_BOOTSTRAP_HIGH ;hw6
```
  0-12   hw6: STATUS (13bit) (extra bits, expanding STATUS in "WL_BOOTSTRAP"?)
```

ATH:0140B8h/01415Ch - ANTENNA_SLEEP_CONTROL/ANTENNA_CONTROL ;hw4/hw6
```
  0-4    hw4: ENABLE   (5bit)          ;\
  5-9    hw4: VALUE    (5bit)          ; hw4 "ANTENNA_SLEEP_CONTROL"
  10-14  hw4: OVERRIDE (5bit)          ;/
  0-3    hw6: ENABLE   (4bit)          ;\
  4-7    hw6: VALUE    (4bit)          ;
  8-11   hw6: OVERRIDE (4bit)          ;
```

```
  12-13  hw6: LED_SEL  (2bit)           ; hw6 "ANTENNA_CONTROL"
  14     hw6: SPI_MODE                  ;
  15     hw6: SPI_CS                    ;
  16     hw6: RX_CLEAR                  ;/

 ____________________________________ hw6 stuff ____________________________________
```

ATH:014168h - AMBA_DEBUG_BUS ;hw6

```
  0-4    SEL
```

ATH:01416Ch - CPU_MBIST ;hw6

```
  0      DONE
  1      GLOBAL_FAIL
  2-10   BLOCK_FAIL

 ____________________________________ hw4 stuff ____________________________________
```

ATH:014094h - FUNC_BUS ;hw4

```
  0-21   OE_L
  22     GPIO_MODE
```

ATH:0140A4h - CLOCK_GPIO ;hw4

```
  0      hw4: BT_CLK_OUT_EN
  1      hw4: BT_CLK_REQ_EN
  2      hw4: CLK_REQ_OUT_EN

 _________________________________ hw2 GPIO PIN config _________________________________
```

ATH:0140D4h - ANTD_PIN - Config: Pad Pull ;hw2
ATH:0140DCh - GPIO_H_PIN - Config: Pad Pull ;hw2
ATH:0140E4h - BT_WLAN_PIN - Config: Pad Pull ;hw2
ATH:0140ECh - CLK32K_PIN - Config: Pad Pull ;hw2

```
  0-1    hw2: PAD_PULL
```

ATH:014070h - SDIO_PIN - Config: Pad Pull/Strength ;hw2
ATH:0140D0h - ANT_PIN - Config: Pad Pull/Strength ;hw2
ATH:0140D8h - GPIO_PIN - Config: Pad Pull/Strength ;hw2
ATH:0140E0h - BT_PIN - Config: Pad Pull/Strength ;hw2
ATH:0140E8h - SI_UART_PIN - Config: Pad Pull/Strength ;hw2

```
  0-1    hw2: PAD_STRENGTH
  2-3    hw2: PAD_PULL
```

ATH:014074h - CLK_REQ_PIN - Config: Pad Pull/Strength/AteOeLow ;hw2

```
  0-1    hw2: PAD_STRENGTH
  2-3    hw2: PAD_PULL
  4      hw2: ATE_OE_L

 ____________________________________ hw2 LA stuff ____________________________________
```

ATH:014088h - LA_CONTROL ;hw2

```
  0      hw2: TRIGGERED
  1      hw2: RUN
```

ATH:01408Ch - LA_CLOCK ;hw2

```
  0-7    hw2: DIV
```

ATH:014090h - LA_STATUS ;hw2

```
  0      hw2: INTERRUPT
```

ATH:014094h - LA_TRIGGER_SAMPLE ;hw2
```
  0-15   hw2: COUNT
```

ATH:014098h - LA_TRIGGER_POSITION ;hw2
```
  0-15   hw2: VALUE
```

ATH:01409Ch - LA_PRE_TRIGGER ;hw2
ATH:0140A0h - LA_POST_TRIGGER ;hw2
```
  0-15   hw2: COUNT
```

ATH:0140A4h - LA_FILTER_CONTROL ;hw2
```
  0      hw2: DELTA
```

ATH:0140A8h - LA_FILTER_DATA ;hw2
ATH:0140B0h - LA_TRIGGERA_DATA ;hw2
ATH:0140B8h - LA_TRIGGERB_DATA ;hw2
ATH:0140ACh - LA_FILTER_WILDCARD ;hw2
ATH:0140B4h - LA_TRIGGERA_WILDCARD ;hw2
ATH:0140BCh - LA_TRIGGERB_WILDCARD ;hw2
```
  0-17   hw2: MATCH                ... maybe related to GPIO_PIN0..17 ?
```

ATH:0140C0h - LA_TRIGGER ;hw2
```
  0-2    hw2: EVENT
```

ATH:0140C4h - LA_FIFO ;hw2
```
  0      hw2: EMPTY
  1      hw2: FULL
```

ATH:0140C8h..0140CCh - LA[0..1] ;hw2
```
  0-17   hw2: DATA
```

DSi Atheros Wifi - Internal I/O - 018000h - MBOX Registers (hw2/hw4/hw6)

These registers are same in hw2/hw4/hw6, except that:
```
  GMBOX registers exist in hw4/hw6 only
  STE_MODE register exists in hw6 only
  WLAN_MBOX_INT_xxx bit18,19 exist in hw6 only
```
And, register names didn't have had the "WLAN_" prefix in hw2.

```
 ____________________________ Manual MBOX Transfer ______________________________
```

ATH:018000h..01800Ch - WLAN_MBOX_FIFO[0..3]
```
  0-7    DATA: DATABYTE
  8-11   DATA: zero?
  12-15  DATA: zero? maybe copy of MBOX_FIFO_STATUS bit12-15 ? (FULL)
  16-19  DATA: looks like copy of MBOX_FIFO_STATUS bit16-19 ?  (EMPTY)
  20-31  -
```
READ: Allows to read incoming MBOX data; before reading this register, the data MUST be manually copied to this register via WLAN_MBOX_TXFIFO_POP[n], then read this register, and check the EMPTY flag; this requires "double indexing" as so: for MBOX(n), test "WLAN_MBOX_FIFO[n].bit(16+n)", if the bit is zero, then bit0-7 contains valid data.
WRITE: Allows to send outgoing MBOX data (write the databyte, with zeroes in bit8-31); before doing so, one SHOULD check if the FIFO is full via WLAN_MBOX_FIFO_STATUS.

ATH:0180F0h..0180FCh - WLAN_MBOX_TXFIFO_POP[0..3]
```
0      DATA  ... uh 4x1bit ?   for MBOX0..3 ?
1-31   -
```
Writing 0 to WLAN_MBOX_TXFIFO_POP[n] does remove the oldest "TXFIFO" entry (the data transmitted from SDIO side to xtensa side via MBOXn), and stores that value (and a copy of the WLAN_MBOX_FIFO_STATUS bits) in WLAN_MBOX_FIFO[n].

ATH:018100h..01810Ch - WLAN_MBOX_RXFIFO_POP[0..3]
```
0      DATA  ... uh 4x1bit ?   for MBOX0..3 ?
1-31   -
```
Probably similar as above, but for opposite direction (ie. allowing to read data that was "sent-to-the-host"; normally such data should be read by the host, so one would use this feature only if one wants to screw up the normal transfer flow).

_____ DMA MBOX Transfer _____

ATH:018014h - WLAN_MBOX_DMA_POLICY
```
0      RX_ORDER
1      RX_QUANTUM
2      TX_ORDER
3      TX_QUANTUM
4-31   -
```

ATH:018018h - WLAN_MBOX0_DMA_RX_DESCRIPTOR_BASE
ATH:018020h - WLAN_MBOX0_DMA_TX_DESCRIPTOR_BASE
ATH:018028h - WLAN_MBOX1_DMA_RX_DESCRIPTOR_BASE
ATH:018030h - WLAN_MBOX1_DMA_TX_DESCRIPTOR_BASE
ATH:018038h - WLAN_MBOX2_DMA_RX_DESCRIPTOR_BASE
ATH:018040h - WLAN_MBOX2_DMA_TX_DESCRIPTOR_BASE
ATH:018048h - WLAN_MBOX3_DMA_RX_DESCRIPTOR_BASE
ATH:018050h - WLAN_MBOX3_DMA_TX_DESCRIPTOR_BASE
ATH:018114h - WLAN_GMBOX0_DMA_RX_DESCRIPTOR_BASE - hw4/hw6 only
ATH:01811Ch - WLAN_GMBOX0_DMA_TX_DESCRIPTOR_BASE - hw4/hw6 only
```
0-1    -
2-27   ADDRESS
28-31  -
```

ATH:01801Ch - WLAN_MBOX0_DMA_RX_CONTROL
ATH:018024h - WLAN_MBOX0_DMA_TX_CONTROL
ATH:01802Ch - WLAN_MBOX1_DMA_RX_CONTROL
ATH:018034h - WLAN_MBOX1_DMA_TX_CONTROL
ATH:01803Ch - WLAN_MBOX2_DMA_RX_CONTROL
ATH:018044h - WLAN_MBOX2_DMA_TX_CONTROL
ATH:01804Ch - WLAN_MBOX3_DMA_RX_CONTROL
ATH:018054h - WLAN_MBOX3_DMA_TX_CONTROL
ATH:018118h - WLAN_GMBOX0_DMA_RX_CONTROL - hw4/hw6 only
ATH:018120h - WLAN_GMBOX0_DMA_TX_CONTROL - hw4/hw6 only
```
0      STOP
1      START
2      RESUME
3-31   -
```

_____ Status _____

ATH:018010h - WLAN_MBOX_FIFO_STATUS
```
0-11   -
```

```
  12-15  FULL flags for MBOX 0..3
  16-19  EMPTY flags for MBOX 0..3
  20-31  -
```

ATH:018058h - WLAN_MBOX_INT_STATUS
ATH:01805Ch - WLAN_MBOX_INT_ENABLE

```
  0-7    HOST           Interrupt 0..7 from Host        ;SDIO 1:00472h.bit0..7
  8-11   RX_NOT_FULL    MBOX0..3 RX FIFO Not Full
  12-15  TX_NOT_EMPTY   MBOX0..3 TX FIFO Not Empty
  16     RX_UNDERFLOW   MBOX RX Underflow (tried to read from empty fifo)
  17     TX_OVERFLOW    MBOX TX Overflow (tried to write to full fifo)
  18     hw6: FRAME_DONE                            ;\hw6.0 only
  19     hw6: NO_RX_MBOX_DATA_AVA                   ;/
  20-23  TX_DMA_COMPLETE      MBOX0..3 TX DMA Complete
  24-27  TX_DMA_EOM_COMPLETE  MBOX0..3 TX DMA Complete .. End of message?
  28-31  RX_DMA_COMPLETE      MBOX0..3 RX DMA Complete
```

ATH:018124h - WLAN_GMBOX_INT_STATUS - hw4/hw6 only
ATH:018128h - WLAN_GMBOX_INT_ENABLE - hw4/hw6 only

```
  0      RX_NOT_FULL
  1      TX_NOT_EMPTY
  2      TX_DMA_COMPLETE
  3      TX_DMA_EOM_COMPLETE
  4      RX_DMA_COMPLETE
  5      RX_UNDERFLOW
  6      TX_OVERFLOW
  7-31   -
```

_____ SDIO Handshake _____

ATH:018060h - WLAN_INT_HOST

```
  0-7    VECTOR   Interrupt 0..7 to Host               ;SDIO 1:00401h.bit0..7
  8-31   -
```

ATH:018080h..01809Ch - WLAN_LOCAL_COUNT[0..7]
ATH:0180A0h..0180BCh - WLAN_COUNT_INC[0..7]

```
  0-7    VALUE (credit counter)                      ;SDIO 1:00420h..00427h
  8-31   -
```

ATH:0180C0h..0180DCh - WLAN_LOCAL_SCRATCH[0..7]

```
  0-7    VALUE (scratch)                             ;SDIO 1:00460h..00467h
  8-31   -
```

ATH:0180E0h - WLAN_USE_LOCAL_BUS

```
  0      PIN_INIT       ;whatever, maybe PCI bus related (non-SDIO) ?
  1-31   -
```

ATH:0180E4h - WLAN_SDIO_CONFIG

```
  0      CCCR_IOR1       ;SDIO Func I/O Ready bit1 ?    ;SDIO 0:00002h.bit1
  1-31   -
```

ATH:01A000h..01BFFCh - WLAN_HOST_IF_WINDOW[0..2047]

```
  0-7    DATA                                        ;SDIO 1:00000h..007FFh
  8-31   -
```

Allows to access the SDIO Host registers via Internal registers, should be done only for testing purposes.

_____ Misc _____

ATH:0180E8h - WLAN_MBOX_DEBUG

```
  0-2    SEL
```

```
3-31   -
```

ATH:0180ECh - WLAN_MBOX_FIFO_RESET

```
0      INIT
1-31   -
```

ATH:018110h - WLAN_SDIO_DEBUG

```
0-3    SEL
4-31   -
```

ATH:01812Ch - STE_MODE - hw6.0 only

```
0      SEL
1-2    PHA_POL
3      SEL_16BIT
4      SWAP
5      RST
6      SPI_CTRL_EN
```

DSi Atheros Wifi - Internal I/O - 01C000h - Analog Intf (hw2)

ATH:01C000h - SYNTH_SYNTH1 ;aka - PHY_ANALOG_SYNTH1

```
0      MONITOR_SYNTHLOCKVCOK
1      MONITOR_VC2LOW
2      MONITOR_VC2HIGH
3      MONITOR_FB_DIV2
4      MONITOR_REF
5      MONITOR_FB
6      PWUP_LOBUF5G_PD
7      PWUP_LOMIX_PD
8      PWUP_LODIV_PD
9      PWUP_VCOBUF_PD
10-12  SEL_VCMONABUS
13     CON_IVCOBUF
14     CON_IVCOREG
15     CON_VDDVCOREG
16     SPARE_PWD
17     SLIDINGIF
18-19  VCOREGBIAS
20-21  VCOREGLEVEL
22     VCOREGBYPASS
23     PWD_LOBUF5G
24     FORCE_LO_ON
25     PWD_LOMIX
26     PWD_LODIV
27     PWD_PRESC
28     PWD_VCO
29     PWD_VCMON
30     PWD_CP
31     PWD_BIAS
```

ATH:01C004h - SYNTH_SYNTH2 ;aka - PHY_ANALOG_SYNTH2 (one part)

```
0-2    SPARE_BITS
3-4    LOOP_CS
5-9    LOOP_RS
10-14  LOOP_CP
15-19  LOOP_3RD_ORDER_R
20-22  VC_LOW_REF
23-25  VC_MID_REF
26-28  VC_HI_REF
29-31  VC_CAL_REF
```

ATH:01C008h - SYNTH_SYNTH3 ;aka - PHY_ANALOG_SYNTH3

0-5	WAIT_VC_CHECK
6-11	WAIT_CAL_LIN
12-17	WAIT_CAL_BIN
18-23	WAIT_PWRUP
24-29	WAIT_SHORTR_PWRUP
30	SEL_CLK_DIV2
31	DIS_CLK_XTAL

ATH:01C00Ch - SYNTH_SYNTH4 ;aka - PHY_ANALOG_SYNTH4

0	FORCE_SHIFTREG
1	LONGSHIFTSEL
2-3	SPARE_MISC
4	SEL_CLKXTAL_EDGE
5	PSCOUNT_FBSEL
6-7	SDM_DITHER
8	SDM_MODE
9	SDM_DISABLE
10	RESET_PRESC
11-12	PRESCSEL
13	PFD_DISABLE
14	PFDDELAY
15-16	REFDIVSEL
17	VCOCAPPULLUP
18-25	VCOCAP_OVR
26	FORCE_VCOCAP
27	FORCE_PINVC
28	SHORTR_UNTIL_LOCKED
29	ALWAYS_SHORTR
30	DIS_LOSTVC
31	DIS_LIN_CAPSEARCH

ATH:01C010h - SYNTH_SYNTH5 ;aka - PHY_ANALOG_SYNTH2 (other part)

0-1	SPARE
2-3	LOBUF5GTUNE_OVR
4	FORCE_LOBUF5GTUNE
5-8	CAPRANGE3
9-12	CAPRANGE2
13-16	CAPRANGE1
17-20	LOOPLEAKCUR
21	CPLOWLK
22	CPSTEERING_EN
23-24	CPBIAS
25-27	SLOPE_IP
28-31	LOOP_IP0

ATH:01C014h - SYNTH_SYNTH6

0-2	SPARE_BIAS
3-4	VCOBUFBIAS
5-7	ICVCO
8-10	ICSPAREB
11-13	ICSPAREA
14-16	ICLOMIX
17-19	ICLODIV
20-22	ICPRESC
23-25	IRSPARE
26-28	IRVCMON
29-31	IRCP

ATH:01C018h - SYNTH_SYNTH7 ;aka "PHY_ANALOG_SYNTH6" (six) on later hw

0-2	SPARE_READ
3-4	LOBUF5GTUNE

5-8	LOOP_IP
9	VC2LOW
10	VC2HIGH
11	RESET_SDM_B
12	RESET_PSCOUNTERS
13	RESET_PFD
14	RESET_RFD
15	SHORT_R
16-23	VCO_CAP_ST
24	PIN_VC
25	SYNTH_LOCK_VC_OK
26	CAP_SEARCH
27-30	SYNTH_SM_STATE
31	SYNTH_ON

ATH:01C01Ch - SYNTH_SYNTH8 ;aka "PHY_ANALOG_SYNTH7" (seven) on later hw

0	FORCE_FRACLSB
1-17	CHANFRAC
18-26	CHANSEL
27	SPARE
28-29	AMODEREFSEL
30	FRACMODE
31	LOADSYNTHCHANNEL

ATH:01C020h - RF5G_RF5G1

0-1	SPARE
2	REGLO_BYPASS5
3	LO5CONTROL
4-6	LO5_ATB_SEL
7	PDREGLO5
8	PDLO5AGC
9	PDQBUF5
10	PDLO5MIX
11	PDLO5DIV
12-14	TX5_ATB_SEL
15-17	OB5
18-20	DB5
21-23	PWDTXPKD
24-26	TUNE_PADRV5
27	PDPAOUT5
28	PDPADRV5
29	PDTXBUF5
30	PDTXMIX5
31	PDTXLO5

ATH:01C024h - RF5G_RF5G2

0-1	SPARE
2-4	TUNE_LO
5	ENABLE_PCA
6-7	LNA5_ATTENMODE
8	REGFE_BYPASS5
9-11	BVGM5
12-14	BCSLNA5
15-17	BRFVGA5
18-20	TUNE_RFVGA5
21	PDREGFE5
22	PDRFVGA5
23	PDCSLNA5
24	PDVGM5
25	PDCMOSLO5
26-28	RX5_ATB_SEL
29-31	AGCLO_B

ATH:01C028h - RF2G_RF2G1

0-4	SPARE
5	SHORTLNA2
6	LOCONTROL
7	SELLNA
8-10	RF_ATB_SEL
11-13	FE_ATB_SEL
14-16	OB
17-19	DB
20-22	BLNA2
23-25	BLNA1BUF
26-28	BLNA1F
29-31	BLNA1

ATH:01C02Ch - RF2G_RF2G2

0-16	SPARE
17	ENABLE_PCB
18	REGLO_BYPASS
19	REGLNA_BYPASS
20	PDTXMIX
21	PDTXLO
22	PDRXLO
23	PDRFGM
24	PDREGLO
25	PDREGLNA
26	PDPAOUT
27	PDPADRV
28	PDDIV
29	PDCSLNA
30	PDCGLNABUF
31	PDCGLNA

ATH:01C030h - TOP_GAIN

0	SPARE
1-2	RX6DBHIQGAIN
3-5	RX1DBLOQGAIN
6-7	RX6DBLOQGAIN
8-10	RFGMGN
11-12	RFVGA5GAIN
13-16	LNAGAIN
17	LNAON
18-20	PAOUT2GN
21-23	PADRVGN
24	PABUF5GN
25-26	TXV2IGAIN
27-29	TX1DBLOQGAIN
30-31	TX6DBLOQGAIN

ATH:01C034h - TOP_TOP

0	FORCE_XPAON
1	INT2GND
2	PAD2GND
3	INTH2PAD
4	INT2PAD
5-7	REVID
8-9	DATAOUTSEL
10	PDBIAS
11	SYNTHON_FORCE
12	SCLKEN_FORCE
13	OSCON
14	PWDCLKIN
15	LOCALXTAL

16	PWDDAC
17	PWDADC
18	PWDPLL
19	LOCALADDAC
20	CALTX
21	PAON
22	TXON
23	RXON
24	SYNTHON
25	BMODE
26	CAL_RESIDUE
27	CALDC
28	CALFC
29	LOCALMODE
30	LOCALRXGAIN
31	LOCALTXGAIN

ATH:01C038h - BIAS_BIAS_SEL

0	PWD_ICLDO25
1-3	PWD_ICTXPC25
4-6	PWD_ICTSENS25
7-9	PWD_ICXTAL25
10-12	PWD_ICCOMPBIAS25
13	PWD_ICCPLL25
14	PWD_ICREFOPAMPBIAS25
15	PWD_IRREFMASTERBIAS12P5
16	PWD_IRDACREGREF12P5
17-19	PWD_ICREFBUFBIAS12P5
20	SPARE
21-24	SEL_SPARE
25-30	SEL_BIAS
31	PADON

ATH:01C03Ch - BIAS_BIAS1

0-1	SPARE
2-4	PWD_IC5GMIXQ25
5-7	PWD_IC5GQB25
8-10	PWD_IC5GTXBUF25
11-13	PWD_IC5GTXPA25
14	PWD_IC5GRXRF25
15	PWD_ICDETECTORA25
16	PWD_ICDETECTORB25
17-19	PWD_IC2GLNAREG25
20-22	PWD_IC2GLOREG25
23-25	PWD_IC2GRFFE25
26-28	PWD_IC2GVGM25
29-31	PWD_ICDAC2BB25

ATH:01C040h - BIAS_BIAS2

0-2	PWD_IR5GRFVREF2525
3-5	PWD_IR2GLNAREG25
6-8	PWD_IR2GLOREG25
9-11	PWD_IR2GTXMIX25
12	PWD_IRLDO25
13-15	PWD_IRTXPC25
16-18	PWD_IRTSENS25
19-21	PWD_IRXTAL25
22	PWD_IRPLL25
23-25	PWD_IC5GLOREG25
26-28	PWD_IC5GDIV25
29-31	PWD_IC5GMIXI25

ATH:01C044h - BIAS_BIAS3

0	SPARE
1-3	PWD_ICDACREG12P5
4-6	PWD_IR25SPARE2
7-9	PWD_IR25SPARE1
10-12	PWD_IC25SPARE2
13-15	PWD_IC25SPARE1
16	PWD_IRBB50
17	PWD_IRSYNTH50
18-20	PWD_IC2GDIV50
21	PWD_ICBB50
22	PWD_ICSYNTH50
23-25	PWD_ICDAC50
26-28	PWD_IR5GAGC25
29-31	PWD_IR5GTXMIX25

ATH:01C048h - TXPC_TXPC

0-1	ATBSEL
2	SELCOUNT
3-4	SELINIT
5	ON1STSYNTHON
6-13	N
14-15	TSMODE
16	SELCMOUT
17	SELMODREF
18	CLKDELAY
19	NEGOUT
20	CURHALF
21	TESTPWDPC
22-27	TESTDAC
28-29	TESTGAIN
30	TEST
31	SELINTPD

ATH:01C04Ch - TXPC_MISC

0-5	SPARE
6-7	XTALDIV
8-17	DECOUT
18-20	SPARE_A
21	SELTSN
22	SELTSP
23	LOCALBIAS2X
24	LOCALBIAS
25	PWDXINPAD
26	PWDCLKIND
27	NOTCXODET
28	LDO_TEST_MODE
29-30	LEVEL
31	FLIPBMODE

ATH:01C050h - RXTXBB_RXTXBB1

0	PDHIQ
1	PDLOQ
2	PDOFFSETI2V
3	PDOFFSETHIQ
4	PDOFFSETLOQ
5	PDRXTXBB
6	PDI2V
7	PDV2I
8	PDDACINTERFACE
6-16	SEL_ATB
17-18	FNOTCH
19-31	SPARE

ATH:01C054h - RXTXBB_RXTXBB2

0	PATH_OVERRIDE
1	PATH1LOQ_EN
2	PATH2LOQ_EN
3	PATH3LOQ_EN
4	PATH1HIQ_EN
5	PATH2HIQ_EN
6	FILTERDOUBLEBW
7	LOCALFILTERTUNING
8-12	FILTERFC
13-14	CMSEL
15	SEL_I2V_TEST
16	SEL_HIQ_TEST
17	SEL_LOQ_TEST
18	SEL_DAC_TEST
19	SELBUFFER
20	SHORTBUFFER
21-22	SPARE
23-25	IBN_37P5_OSI2V_CTRL
26-28	IBN_37P5_OSLO_CTRL
29-31	IBN_37P5_OSHI_CTRL

ATH:01C058h - RXTXBB_RXTXBB3

0-2	IBN_100U_TEST_CTRL
3-5	IBRN_12P5_CM_CTRL
6-8	IBN_25U_LO2_CTRL
9-11	IBN_25U_LO1_CTRL
12-14	IBN_25U_HI2_CTRL
15-17	IBN_25U_HI1_CTRL
18-20	IBN_25U_I2V_CTRL
21-23	IBN_25U_BKV2I_CTRL
24-26	IBN_25U_CM_BUFAMP_CTRL
27-31	SPARE

ATH:01C05Ch - RXTXBB_RXTXBB4

0-4	OFSTCORRI2VQ
5-9	OFSTCORRI2VI
10-14	OFSTCORRLOQ
15-19	OFSTCORRLOI
20-24	OFSTCORRHIQ
25-29	OFSTCORRHII
30	LOCALOFFSET
31	SPARE

ATH:01C060h - ADDAC_ADDAC1 ;aka "A/D and D/A Converter"

0-5	SPARE
6	DISABLE_DAC_REG
7-8	CM_SEL
9	INV_CLK160_ADC
10	SELMANPWDS
11	FORCEMSBLOW
12	PWDDAC
13	PWDADC
14	PWDPLL
15-22	PLL_FILTER
23-25	PLL_ICP
26-27	PLL_ATB
28-30	PLL_SCLAMP
31	PLL_SVREG

ATH:01C080h - SW_OVERRIDE

0	ENABLE

```
1       SUPDATE_DELAY
```

ATH:01C084h - SIN_VAL
```
0       SIN
```

ATH:01C088h - SW_SCLK
```
0       SW_SCLK
```

ATH:01C08Ch - SW_CNTL
```
0       SW_SOUT
1       SW_SUPDATE
2       SW_SCAPTURE
```

DSi Atheros Wifi - Internal I/O - 01C000h - Analog Intf (hw4/hw6)

These registers are same in hw4/hw6, except for some small differences:
```
0001C050h  one new bit in hw6.0
0001C148h  several new bits in hw6.0
0001C740h  added/removed/renumbered bits in hw6.0
0001C744h  two changed/renamed bits in hw6.0
```

ATH:01C000h - PHY_ANALOG_RXRF_BIAS1
```
0       SPARE
1-3     PWD_IR25SPARE
4-6     PWD_IR25LO18
7-9     PWD_IC25LO36
10-12   PWD_IC25MXR2_5GH
13-15   PWD_IC25MXR5GH
16-18   PWD_IC25VGA5G
19-21   PWD_IC75LNA5G
22-24   PWD_IR25LO24
25-27   PWD_IC25MXR2GH
28-30   PWD_IC75LNA2G
31      PWD_BIAS
```

ATH:01C004h - PHY_ANALOG_RXRF_BIAS2
```
0       SPARE
1-3     PKEN
4-6     VCMVALUE
7       PWD_VCMBUF
8-10    PWD_IR25SPAREH
11-13   PWD_IR25SPARE
14-16   PWD_IC25LNABUF
17-19   PWD_IR25AGCH
20-22   PWD_IR25AGC
23-25   PWD_IC25AGC
26-28   PWD_IC25VCMBUF
29-31   PWD_IR25VCM
```

ATH:01C008h - PHY_ANALOG_RXRF_GAINSTAGES
```
0       SPARE
1       LNAON_CALDC
2-3     VGA5G_CAP
4-5     LNA5G_CAP
6       LNA5G_SHORTINP
7       PWD_LO5G
8       PWD_VGA5G
9       PWD_MXR5G
10      PWD_LNA5G
11-12   LNA2G_CAP
```

13	LNA2G_SHORTINP
14	LNA2G_LP
15	PWD_LO2G
16	PWD_MXR2G
17	PWD_LNA2G
18-19	MXR5G_GAIN_OVR
20-22	VGA5G_GAIN_OVR
23-25	LNA5G_GAIN_OVR
26-27	MXR2G_GAIN_OVR
28-30	LNA2G_GAIN_OVR
31	RX_OVERRIDE

ATH:01C00Ch - PHY_ANALOG_RXRF_AGC

0	RF5G_ON_DURING_CALPA
1	RF2G_ON_DURING_CALPA
2	AGC_OUT (R)
3	LNABUFGAIN2X
4	LNABUF_PWD_OVR
5	PWD_LNABUF
6-8	AGC_FALL_CTRL
9-14	AGC5G_CALDAC_OVR
15-18	AGC5G_DBDAC_OVR
19-24	AGC2G_CALDAC_OVR
25-28	AGC2G_DBDAC_OVR
29	AGC_CAL_OVR
30	AGC_ON_OVR
31	AGC_OVERRIDE

ATH:01C040h - PHY_ANALOG_TXRF1

0	PDLOBUF5G
1	PDLODIV5G
2	LOBUF5GFORCED
3	LODIV5GFORCED
4-7	PADRV2GN5G
8-11	PADRV3GN5G
12-15	PADRV4GN5G
16	LOCALTXGAIN5G
17	PDOUT2G
18	PDDR2G
19	PDMXR2G
20	PDLOBUF2G
21	PDLODIV2G
22	LOBUF2GFORCED
23	LODIV2GFORCED
24-30	PADRVGN2G
31	LOCALTXGAIN2G

ATH:01C044h - PHY_ANALOG_TXRF2

0-2	D3B5G
3-5	D4B5G
6-8	OCAS2G
9-11	DCAS2G
12-14	OB2G_PALOFF
15-17	OB2G_QAM
18-20	OB2G_PSK
21-23	OB2G_CCK
24-26	DB2G
27-30	PDOUT5G
31	PDMXR5G

ATH:01C048h - PHY_ANALOG_TXRF3

0-1	FILTR2G
2	PWDFB2_2G

```
3      PWDFB1_2G
4      PDFB2G
5-6    RDIV5G
7-9    CAPDIV5G
10     PDPREDIST5G
11-12  RDIV2G
13     PDPREDIST2G
14-16  OCAS5G
17-19  D2CAS5G
20-22  D3CAS5G
23-25  D4CAS5G
26-28  OB5G
29-31  D2B5G
```

ATH:01C04Ch - PHY_ANALOG_TXRF4

```
0-1    PK1B2G_CCK
2-4    MIOB2G_QAM
5-7    MIOB2G_PSK
8-10   MIOB2G_CCK
11-13  COMP2G_QAM
14-16  COMP2G_PSK
17-19  COMP2G_CCK
20-22  AMP2B2G_QAM
23-25  AMP2B2G_PSK
26-28  AMP2B2G_CCK
29-31  AMP2CAS2G
```

ATH:01C050h - PHY_ANALOG_TXRF5

```
0      hw4: SPARE5
0      hw6: TXMODPALONLY      ;-hw6.0 only
1      PAL_LOCKED                    (R)
2      FBHI2G                        (R)
3      FBLO2G                        (R)
4      NOPALGAIN2G
5      ENPACAL2G
6-12   OFFSET2G
13     ENOFFSETCAL2G
14-16  REFHI2G
17-19  REFLO2G
20-21  PALCLAMP2G
22-23  PK2B2G_QAM
24-25  PK2B2G_PSK
26-27  PK2B2G_CCK
28-29  PK1B2G_QAM
30-31  PK1B2G_PSK
```

ATH:01C054h - PHY_ANALOG_TXRF6

```
0      PALCLKGATE2G
1-8    PALFLUCTCOUNT2G
9-10   PALFLUCTGAIN2G
11     PALNOFLUCT2G
12-14  GAINSTEP2G
15     USE_GAIN_DELTA2G
16-19  CAPDIV_I2G
20-23  PADRVGN_INDEX_I2G
24-26  VCMONDELAY2G
27-30  CAPDIV2G
31     CAPDIV2GOVR
```

ATH:01C058h - PHY_ANALOG_TXRF7 ;PADRVGNTAB_0..4

```
0-1    SPARE7
2-7    PADRVGNTAB_4
8-13   PADRVGNTAB_3
```

14-19	PADRVGNTAB_2	
20-25	PADRVGNTAB_1	
26-31	PADRVGNTAB_0	

ATH:01C05Ch - PHY_ANALOG_TXRF8 ;PADRVGNTAB_5..9

0-1	SPARE8	
2-7	PADRVGNTAB_9	
8-13	PADRVGNTAB_8	
14-19	PADRVGNTAB_7	
20-25	PADRVGNTAB_6	
26-31	PADRVGNTAB_5	

ATH:01C060h - PHY_ANALOG_TXRF9 ;PADRVGNTAB_10..14

0-1	SPARE9	
2-7	PADRVGNTAB_14	
8-13	PADRVGNTAB_13	
14-19	PADRVGNTAB_12	
20-25	PADRVGNTAB_11	
26-31	PADRVGNTAB_10	

ATH:01C064h - PHY_ANALOG_TXRF10

0-2	SPARE10	
3	PDOUT5G_3CALTX	
4-6	D3B5GCALTX	
7-9	D4B5GCALTX	
10-16	PADRVGN2GCALTX	
17-19	DB2GCALTX	
20	CALTXSHIFT	
21	CALTXSHIFTOVR	
22-27	PADRVGN2G_SMOUT	(R)
28-31	PADRVGN_INDEX2G_SMOUT	(R)

ATH:01C068h - PHY_ANALOG_TXRF11

0-1	SPARE11	
2-4	PWD_IR25MIXDIV5G	
5-7	PWD_IR25PA2G	
8-10	PWD_IR25MIXBIAS2G	
11-13	PWD_IR25MIXDIV2G	
14-16	PWD_ICSPARE	
17-19	PWD_IC25TEMPSEN	
20-22	PWD_IC25PA5G2	
23-25	PWD_IC25PA5G1	
26-28	PWD_IC25MIXBUF5G	
29-31	PWD_IC25PA2G	

ATH:01C06Ch - PHY_ANALOG_TXRF12

0-7	SPARE12_2	(R)
8-9	SPARE12_1	
10-13	ATBSEL5G	
14-16	ATBSEL2G	
17-19	PWD_IRSPARE	
20-22	PWD_IR25TEMPSEN	
23-25	PWD_IR25PA5G2	
26-28	PWD_IR25PA5G1	
29-31	PWD_IR25MIXBIAS5G	

ATH:01C080h - PHY_ANALOG_SYNTH1

0-2	SEL_VCMONABUS	
3-5	SEL_VCOABUS	
6	MONITOR_SYNTHLOCKVCOK	
7	MONITOR_VC2LOW	
8	MONITOR_VC2HIGH	

9	MONITOR_FB_DIV2
10	MONITOR_REF
11	MONITOR_FB
12	SEVENBITVCOCAP
13-15	PWUP_PD
16	PWD_VCOBUF
17-18	VCOBUFGAIN
19-20	VCOREGLEVEL
21	VCOREGBYPASS
22	PWUP_LOREF
23	PWD_LOMIX
24	PWD_LODIV
25	PWD_LOBUF5G
26	PWD_LOBUF2G
27	PWD_PRESC
28	PWD_VCO
29	PWD_VCMON
30	PWD_CP
31	PWD_BIAS

ATH:01C084h - PHY_ANALOG_SYNTH2

0-3	CAPRANGE3
4-7	CAPRANGE2
8-11	CAPRANGE1
12-15	LOOPLEAKCUR_INTN
16	CPLOWLK_INTN
17	CPSTEERING_EN_INTN
18-19	CPBIAS_INTN
20-22	VC_LOW_REF
23-25	VC_MID_REF
26-28	VC_HI_REF
29-31	VC_CAL_REF

ATH:01C088h - PHY_ANALOG_SYNTH3

0-5	WAIT_VC_CHECK
6-11	WAIT_CAL_LIN
12-17	WAIT_CAL_BIN
18-23	WAIT_PWRUP
24-29	WAIT_SHORTR_PWRUP
30	SEL_CLK_DIV2
31	DIS_CLK_XTAL

ATH:01C08Ch - PHY_ANALOG_SYNTH4

0	PS_SINGLE_PULSE
1	LONGSHIFTSEL
2-3	LOBUF5GTUNE_OVR
4	FORCE_LOBUF5GTUNE
5	PSCOUNT_FBSEL
6-7	SDM_DITHER1
8	SDM_MODE
9	SDM_DISABLE
10	RESET_PRESC
11-12	PRESCSEL
13	PFD_DISABLE
14	PFDDELAY_FRACN
15	FORCE_LO_ON
16	CLKXTAL_EDGE_SEL
17	VCOCAPPULLUP
18-25	VCOCAP_OVR
26	FORCE_VCOCAP
27	FORCE_PINVC
28	SHORTR_UNTIL_LOCKED
29	ALWAYS_SHORTR

30	DIS_LOSTVC	
31	DIS_LIN_CAPSEARCH	

ATH:01C090h - PHY_ANALOG_SYNTH5

0-1	VCOBIAS	
2-4	PWDB_ICLOBUF5G50	
5-7	PWDB_ICLOBUF2G50	
8-10	PWDB_ICVCO25	
11-13	PWDB_ICVCOREG25	
14	PWDB_IRVCOREG50	
15-17	PWDB_ICLOMIX	
18-20	PWDB_ICLODIV50	
21-23	PWDB_ICPRESC50	
24-26	PWDB_IRVCMON25	
27-29	PWDB_IRPFDCP	
30-31	SDM_DITHER2	

ATH:01C094h - PHY_ANALOG_SYNTH6

0-1	LOBUF5GTUNE	(R)
2-8	LOOP_IP	(R)
9	VC2LOW	(R)
10	VC2HIGH	(R)
11	RESET_SDM_B	(R)
12	RESET_PSCOUNTERS	(R)
13	RESET_PFD	(R)
14	RESET_RFD	(R)
15	SHORT_R	(R)
16-23	VCO_CAP_ST	(R)
24	PIN_VC	(R)
25	SYNTH_LOCK_VC_OK	(R)
26	CAP_SEARCH	(R)
27-30	SYNTH_SM_STATE	(R)
31	SYNTH_ON	(R)

ATH:01C098h - PHY_ANALOG_SYNTH7

0	OVRCHANDECODER	
1	FORCE_FRACLSB	
2-18	CHANFRAC	
19-27	CHANSEL	
28-29	AMODEREFSEL	
30	FRACMODE	
31	LOADSYNTHCHANNEL	

ATH:01C09Ch - PHY_ANALOG_SYNTH8

0	CPSTEERING_EN_FRACN	
1-7	LOOP_ICPB	
8-11	LOOP_CSB	
12-16	LOOP_RSB	
17-21	LOOP_CPB	
22-26	LOOP_3RD_ORDER_RB	
27-31	REFDIVB	

ATH:01C0A0h - PHY_ANALOG_SYNTH9

0	PFDDELAY_INTN	
1-3	SLOPE_ICPA0	
4-7	LOOP_ICPA0	
8-11	LOOP_CSA0	
12-16	LOOP_RSA0	
17-21	LOOP_CPA0	
22-26	LOOP_3RD_ORDER_RA	
27-31	REFDIVA	

ATH:01C0A4h - PHY_ANALOG_SYNTH10

0-1	SPARE10A
2-4	PWDB_ICLOBIAS50
5-7	PWDB_IRSPARE25
8-10	PWDB_ICSPARE25
11-13	SLOPE_ICPA1
14-17	LOOP_ICPA1
18-21	LOOP_CSA1
22-26	LOOP_RSA1
27-31	LOOP_CPA1

ATH:01C0A8h - PHY_ANALOG_SYNTH11

0-4	SPARE11A
5	FORCE_LOBUF5G_ON
6-7	LOREFSEL
8-9	LOBUF2GTUNE
10	CPSTEERING_MODE
11-13	SLOPE_ICPA2
14-17	LOOP_ICPA2
18-21	LOOP_CSA2
22-26	LOOP_RSA2
27-31	LOOP_CPA2

ATH:01C0ACh - PHY_ANALOG_SYNTH12

0-9	SPARE12A
10-13	LOOPLEAKCUR_FRACN
14	CPLOWLK_FRACN
15-16	CPBIAS_FRACN
17	SYNTHDIGOUTEN
18	STRCONT
19-22	VREFMUL3
23-26	VREFMUL2
27-30	VREFMUL1
31	CLK_DOUBLER_EN

ATH:01C0B0h - PHY_ANALOG_SYNTH13

0	SPARE13A
1-3	SLOPE_ICPA_FRACN
4-7	LOOP_ICPA_FRACN
8-11	LOOP_CSA_FRACN
12-16	LOOP_RSA_FRACN
17-21	LOOP_CPA_FRACN
22-26	LOOP_3RD_ORDER_RA_FRACN
27-31	REFDIVA_FRACN

ATH:01C0B4h - PHY_ANALOG_SYNTH14

0-1	SPARE14A
2-3	LOBUF5GTUNE_3
4-5	LOBUF2GTUNE_3
6-7	LOBUF5GTUNE_2
8-9	LOBUF2GTUNE_2
10	PWD_LOBUF5G_3
11	PWD_LOBUF2G_3
12	PWD_LOBUF5G_2
13	PWD_LOBUF2G_2
14-16	PWUPLO23_PD
17-19	PWDB_ICLOBUF5G50_3
20-22	PWDB_ICLOBUF2G50_3
23-25	PWDB_ICLOBUF5G50_2
26-28	PWDB_ICLOBUF2G50_2
29-31	PWDB_ICLVLSHFT

ATH:01C0C0h - PHY_ANALOG_BIAS1

0-6	SPARE1
7-9	PWD_IC25V2IQ
10-12	PWD_IC25V2II
13-15	PWD_IC25BB
16-18	PWD_IC25DAC
19-21	PWD_IC25FIR
22-24	PWD_IC25ADC
25-31	BIAS_SEL

ATH:01C0C4h - PHY_ANALOG_BIAS2

0-4	SPARE2
5-7	PWD_IC25XPA
8-10	PWD_IC25XTAL
11-13	PWD_IC25TXRF
14-16	PWD_IC25RXRF
17-19	PWD_IC25SYNTH
20-22	PWD_IC25PLLREG
23-25	PWD_IC25PLLCP2
26-28	PWD_IC25PLLCP
29-31	PWD_IC25PLLGM

ATH:01C0C8h - PHY_ANALOG_BIAS3

0-1	SPARE3
2-4	PWD_IR25SAR
5-7	PWD_IR25TXRF
8-10	PWD_IR25RXRF
11-13	PWD_IR25SYNTH
14-16	PWD_IR25PLLREG
17-19	PWD_IR25BB
20-22	PWD_IR50DAC
23-25	PWD_IR25DAC
26-28	PWD_IR25FIR
29-31	PWD_IR50ADC

ATH:01C0CCh - PHY_ANALOG_BIAS4

0-10	SPARE4
11-13	PWD_IR25SPARED
14-16	PWD_IR25SPAREC
17-19	PWD_IR25SPAREB
20-22	PWD_IR25XPA
23-25	PWD_IC25SPAREC
26-28	PWD_IC25SPAREB
29-31	PWD_IC25SPAREA

ATH:01C100h - PHY_ANALOG_RXTX1

0	SCFIR_GAIN
1	MANRXGAIN
2-5	AGC_DBDAC
6	OVR_AGC_DBDAC
7	ENABLE_PAL
8	ENABLE_PAL_OVR
9-11	TX1DB_BIQUAD
12-13	TX6DB_BIQUAD
14	PADRVHALFGN2G
15-18	PADRV2GN
19-22	PADRV3GN5G
23-26	PADRV4GN5G
27-30	TXBB_GC
31	MANTXGAIN

ATH:01C104h - PHY_ANALOG_RXTX2

0	BMODE
1	BMODE_OVR
2	SYNTHON
3	SYNTHON_OVR
4-5	BW_ST
6	BW_ST_OVR
7	TXON
8	TXON_OVR
9	PAON
10	PAON_OVR
11	RXON
12	RXON_OVR
13	AGCON
14	AGCON_OVR
15-17	TXMOD
18	TXMOD_OVR
19-21	RX1DB_BIQUAD
22-23	RX6DB_BIQUAD
24-25	MXRGAIN
26-28	VGAGAIN
29-31	LNAGAIN

ATH:01C108h - PHY_ANALOG_RXTX3

0-2	SPARE3
3	SPURON
4	PAL_LOCKEDEN
5	DACFULLSCALE
6	ADCSHORT
7	DACPWD
8	DACPWD_OVR
9	ADCPWD
10	ADCPWD_OVR
11-16	AGC_CALDAC
17	AGC_CAL
18	AGC_CAL_OVR
19	LOFORCEDON
20	CALRESIDUE
21	CALRESIDUE_OVR
22	CALFC
23	CALFC_OVR
24	CALTX
25	CALTX_OVR
26	CALTXSHIFT
27	CALTXSHIFT_OVR
28	CALPA
29	CALPA_OVR
30	TURBOADC
31	TURBOADC_OVR

ATH:01C140h - PHY_ANALOG_BB1

0	I2V_CURR2X
1	ENABLE_LOQ
2	FORCE_LOQ
3	ENABLE_NOTCH
4	FORCE_NOTCH
5	ENABLE_BIQUAD
6	FORCE_BIQUAD
7	ENABLE_OSDAC
8	FORCE_OSDAC
9	ENABLE_V2I
10	FORCE_V2I
11	ENABLE_I2V

```
12     FORCE_I2V
13-15  CMSEL
16-17  ATBSEL
18     PD_OSDAC_CALTX_CALPA
19-23  OFSTCORRI2VQ
24-28  OFSTCORRI2VI
29     LOCALOFFSET
30-31  RANGE_OSDAC
```

ATH:01C144h - PHY_ANALOG_BB2

```
0-3    SPARE
4-7    MXR_HIGHGAINMASK
8-9    SEL_TEST
10-14  RCFILTER_CAP
15     OVERRIDE_RCFILTER_CAP
16-19  FNOTCH
20     OVERRIDE_FNOTCH
21-25  FILTERFC
26     OVERRIDE_FILTERFC
27     I2V2RXOUT_EN
28     BQ2RXOUT_EN
29     RXIN2I2V_EN
30     RXIN2BQ_EN
31     SWITCH_OVERRIDE
```

ATH:01C148h - PHY_ANALOG_BB3

```
0-7    SPARE
8-15   hw4: SPARE
8-9    hw6: SEL_OFST_READBK           ;\
10     hw6: OVERRIDE_RXONLY_FILTERFC  ; hw6.0 only
11-15  hw6: RXONLY_FILTERFC           ;/
16-20  FILTERFC                       (R)
21-25  OFSTCORRI2VQ                   (R)
26-30  OFSTCORRI2VI                   (R)
31     EN_TXBBCONSTCUR
```

ATH:01C280h - PHY_ANALOG_PLLCLKMODA

```
0      PWD_PLLSDM
1      PWDPLL
2-16   PLLFRAC
17-20  REFDIV
21-30  DIV
31     LOCAL_PLL
```

ATH:01C284h - PHY_ANALOG_PLLCLKMODA2

```
0-3    SPARE
4      DACPWD
5      ADCPWD
6      LOCAL_ADDAC
7-8    DAC_CLK_SEL
9-12   ADC_CLK_SEL
13     LOCAL_CLKMODA
14     PLLBYPASS
15     LOCAL_PLLBYPASS
16-17  PLLATB
18     PLL_SVREG
19     HI_FREQ_EN
20     RST_WARM_INT_L
21     RST_WARM_OVR
22-23  PLL_KVCO
24-26  PLLICP
27-31  PLLFILTER
```

ATH:01C288h - PHY_ANALOG_TOP

0-2	SPARE	
3	PWDBIAS	
4	FLIP_XPABIAS	
5	XPAON2	
6	XPAON5	
7	XPASHORT2GND	
8-11	XPABIASLVL	
12	XPABIAS_EN	
13	ATBSELECT	
14	LOCAL_XPA	
15	XPABIAS_BYPASS	
16	TEST_PADQ_EN	
17	TEST_PADI_EN	
18	TESTIQ_RSEL	
19	TESTIQ_BUFEN	
20	PAD2GND	
21	INTH2PAD	
22	INTH2GND	
23	INT2PAD	
24	INT2GND	
25	PWDPALCLK	
26	INV_CLK320_ADC	
27	FLIP_REFCLK40	
28	FLIP_PLLCLK320	
29	FLIP_PLLCLK160	
30-31	CLK_SEL	

ATH:01C28Ch - PHY_ANALOG_THERM

0-2	LOREG_LVL	
3-5	RFREG_LVL	
6	SAR_ADC_DONE	(R)
7-14	SAR_ADC_OUT	(R)
15-22	SAR_DACTEST_CODE	
23	SAR_DACTEST_EN	
24	SAR_ADCCAL_EN	
25-26	THERMSEL	
27	SAR_SLOW_EN	
28	THERMSTART	
29	SAR_AUTOPWD_EN	
30	THERMON	
31	LOCAL_THERM	

ATH:01C290h - PHY_ANALOG_XTAL

0-5	SPARE	
6	XTAL_NOTCXODET	
7	LOCALBIAS2X	
8	LOCAL_XTAL	
9	XTAL_PWDCLKIN	
10	XTAL_OSCON	
11	XTAL_PWDCLKD	
12	XTAL_LOCALBIAS	
13	XTAL_SHRTXIN	
14-15	XTAL_DRVSTR	
16-22	XTAL_CAPOUTDAC	
23-29	XTAL_CAPINDAC	
30	XTAL_BIAS2X	
31	TCXODET	(R)

ATH:01C380h - PHY_ANALOG_RBIST_CNTRL

0	ATE_TONEGEN_DC_ENABLE	
1	ATE_TONEGEN_TONE0_ENABLE	

```
2       ATE_TONEGEN_TONE1_ENABLE
3       ATE_TONEGEN_LFTONE0_ENABLE
4       ATE_TONEGEN_LINRAMP_ENABLE_I
5       ATE_TONEGEN_LINRAMP_ENABLE_Q
6       ATE_TONEGEN_PRBS_ENABLE_I
7       ATE_TONEGEN_PRBS_ENABLE_Q
8       ATE_CMAC_DC_WRITE_TO_CANCEL
9       ATE_CMAC_DC_ENABLE
10      ATE_CMAC_CORR_ENABLE
11      ATE_CMAC_POWER_ENABLE
12      ATE_CMAC_IQ_ENABLE
13      ATE_CMAC_I2Q2_ENABLE
14      ATE_CMAC_POWER_HPF_ENABLE
15      ATE_RXDAC_CALIBRATE
16      ATE_RBIST_ENABLE
17      ATE_ADC_CLK_INVERT              ;-newer revision only
```

ATH:01C384h - PHY_ANALOG_TX_DC_OFFSET

```
0-10    ATE_TONEGEN_DC_I
16-26   ATE_TONEGEN_DC_Q
```

ATH:01C388h - PHY_ANALOG_TX_TONEGEN0
ATH:01C38Ch - PHY_ANALOG_TX_TONEGEN1
ATH:01C390h - PHY_ANALOG_TX_LFTONEGEN0

```
0-6     ATE_TONEGEN_TONE_FREQ
8-11    ATE_TONEGEN_TONE_A_EXP
16-23   ATE_TONEGEN_TONE_A_MAN
24-30   ATE_TONEGEN_TONE_TAU_K
```

ATH:01C394h - PHY_ANALOG_TX_LINEAR_RAMP_I
ATH:01C398h - PHY_ANALOG_TX_LINEAR_RAMP_Q

```
0-10    ATE_TONEGEN_LINRAMP_INIT
12-21   ATE_TONEGEN_LINRAMP_DWELL
24-29   ATE_TONEGEN_LINRAMP_STEP
```

ATH:01C39Ch - PHY_ANALOG_TX_PRBS_MAG

```
0-9     ATE_TONEGEN_PRBS_MAGNITUDE_I
16-25   ATE_TONEGEN_PRBS_MAGNITUDE_Q
```

ATH:01C3A0h - PHY_ANALOG_TX_PRBS_SEED_I

```
0-30    ATE_TONEGEN_PRBS_SEED
```

ATH:01C3A4h - PHY_ANALOG_TX_PRBS_SEED_Q

```
0-30    ATE_TONEGEN_PRBS_SEED
```

ATH:01C3A8h - PHY_ANALOG_CMAC_DC_CANCEL

```
0-9     ATE_CMAC_DC_CANCEL_I
16-25   ATE_CMAC_DC_CANCEL_Q
```

ATH:01C3ACh - PHY_ANALOG_CMAC_DC_OFFSET

```
0-3     ATE_CMAC_DC_CYCLES
```

ATH:01C3B0h - PHY_ANALOG_CMAC_CORR

```
0-4     ATE_CMAC_CORR_CYCLES
8-13    ATE_CMAC_CORR_FREQ
```

ATH:01C3B4h - PHY_ANALOG_CMAC_POWER

```
0-3     ATE_CMAC_POWER_CYCLES
```

ATH:01C3B8h - PHY_ANALOG_CMAC_CROSS_CORR

```
  0-3    ATE_CMAC_IQ_CYCLES
```

ATH:01C3BCh - PHY_ANALOG_CMAC_I2Q2
```
  0-3    ATE_CMAC_I2Q2_CYCLES
```

ATH:01C3C0h - PHY_ANALOG_CMAC_POWER_HPF
```
  0-3    ATE_CMAC_POWER_HPF_CYCLES
  4-7    ATE_CMAC_POWER_HPF_WAIT
```

ATH:01C3C4h - PHY_ANALOG_RXDAC_SET1
```
  0-1    ATE_RXDAC_MUX
  4      ATE_RXDAC_HI_GAIN
  8-13   ATE_RXDAC_CAL_WAIT
  16-19  ATE_RXDAC_CAL_MEASURE_TIME
```

ATH:01C3C8h - PHY_ANALOG_RXDAC_SET2
```
  0-4    ATE_RXDAC_I_HI
  8-12   ATE_RXDAC_Q_HI
  16-20  ATE_RXDAC_I_LOW
  24-28  ATE_RXDAC_Q_LOW
```

ATH:01C3CCh - PHY_ANALOG_RXDAC_LONG_SHIFT
```
  0-4    ATE_RXDAC_I_STATIC
  8-12   ATE_RXDAC_Q_STATIC
```

ATH:01C3D0h - PHY_ANALOG_CMAC_RESULTS_I
```
  0-31   ATE_CMAC_RESULTS
```

ATH:01C3D4h - PHY_ANALOG_CMAC_RESULTS_Q
```
  0-31   ATE_CMAC_RESULTS
```

ATH:01C740h - PHY_ANALOG_PMU1

This register differs in hw4/hw6 (in hw6, bits are renumbered, new OVERRIDE bits are added, and the SREG_LVLCTR bit is removed):

```
  hw4    hw6    name
  0-10   0-3    SPARE                       ;-unused
  11     4      OTP_V25_PWD                 ;-OTP V25
  12     5      PAREGON_MAN                 ;\PA REG
  -      6      PAREGON_OVERRIDE_EN         ;/
  13     7      OTPREGON_MAN                ;\OTP REG
  -      8      OTPREGON_OVERRIDE_EN        ;/
  14     9      DREGON_MAN                  ;\DREG
  -      10     DREGON_OVERRIDE_EN          ;/
  15     11     DISCONTMODEEN               ;\DISCONT MODE
  -      12     SWREGDISCONT_OVERRIDE_EN    ;/
  16     13     SWREGON_MAN                 ;\
  -      14     SWREGON_OVERRIDE_EN         ;
  17-18  15-16  SWREG_FREQCUR               ;
  19-21  17-19  SWREG_FREQCAP               ; SW REG
  -      20     SWREGFREQ_OVERRIDE_EN       ;
  22-23  21-22  SWREG_LVLCTR                ;
  -      23     SWREGLVL_OVERRIDE_EN        ;/
  24-25  -      hw4:SREG_LVLCTR             ;-SREG  ;<---- removed in hw6 (!)
  26-27  24-25  DREG_LVLCTR                 ;\DREG
  -      26     DREGLVL_OVERRIDE_EN         ;/
  28     27     PAREG_XPNP                  ;\
  29-31  28-30  PAREG_LVLCTR                ; PA REG
  -      31     PAREGLVL_OVERRIDE_EN        ;/
```

ATH:01C744h - PHY_ANALOG_PMU2
```
  0-7    SPARE
```

```
8      VBATT_1_3TOATB
9      VBATT_1_2TOATB
10     VBATT_2_3TOATB
11     PWD_BANDGAP_MAN
12     PWD_LFO_MAN
13     VBATT_LT_3P2
14     VBATT_LT_2P8
15     VBATT_GT_4P2
16     hw4: PMU_MAN_OVERRIDE_EN              ;\changed/renamed in hw4/hw6
16     hw6: PMU_XPNP_OVERRIDE_EN             ;/
17-18  VBATT_GT_LVLCTR
19     SWREGVSSL2ATB
20-21  SWREGVSSL_LVLCTR
22     SWREGVDDH2ATB
23-24  SWREGVDDH_LVLCTR
25-27  SWREG2ATB
28     OTPREG2ATB
29-30  OTPREG_LVLCTR
31     hw4: DREG_LVLCTR_MANOVR_EN            ;\changed/renamed in hw4/hw6
31     hw6: OTPREG_LVLCTR_MANOVR_EN          ;/
```

DSi Atheros Wifi - Internal I/O - 020000h - WMAC DMA (hw4/hw6)

ATH:020008h - MAC_DMA_CR - MAC Control Register

```
0-1    -
2      hw4: Receive enable (RXE)           (R)       ;\one bit in hw4,
2      hw6: Receive LP enable (RXE_LP)     (R)       ; two bits in hw6
3      hw6: Receive HP enable (RXE_HP)     (R)       ;/
4      -
5      Receive disable (RXD)
6      One-shot software interrupt (SWI)   (R)
```

ATH:02000Ch - MAC_DMA_RXDP - MAC receive queue descriptor pointer ;hw4 only

```
..     Pointer                          <------------ HW4 ONLY
```

ATH:020014h - MAC_DMA_CFG - MAC configuration and status register

```
0      Byteswap TX descriptor words (BE_MODE_XMIT_DESC)
1      Byteswap TX data buffer words (BE_MODE_XMIT_DATA)
2      Byteswap RX descriptor words (BE_MODE_RCV_DESC)
3      Byteswap RX data buffer words (BE_MODE_RCV_DATA)
4      Byteswap register access data words (BE_MODE_MMR)
5      AP/adhoc indication (ADHOC) (0=AP, 1=Adhoc)
6-7    -
8      PHY OK status (PHY_OK)                  (R)
9      hw6: EEPROM_BUSY                        (R)  ;-hw6 only
10     Clock gating disable (CLKGATE_DIS)
11     hw6: HALT_REQ                                ;\
12     hw6: HALT_ACK                           (R)  ;
13-16  -                                            ;
17-18  hw6: REQ_Q_FULL_THRESHOLD                    ; hw6 only
19     hw6: MISSING_TX_INTR_FIX_ENABLE              ;
20     hw6: LEGACY_INT_MIT_MODE_ENABLE              ;
21     hw6: RESET_INT_MIT_CNTRS                     ;/
```

ATH:020018h - MAC_DMA_RXBUFPTR_THRESH ;hw6 only

```
0-3    hw6: HP_DATA                                 ;\hw6 only
8-14   hw6: LP_DATA                                 ;/
```

ATH:02001Ch - MAC_DMA_TXDPPTR_THRESH ;hw6 only

```
0-3    DATA                                         ;-hw6 only
```

ATH:020020h - MAC_DMA_MIRT - Maximum rate threshold register

```
0-15   Threshold (RATE_THRESH)
```

ATH:020024h - MAC_DMA_IER aka MAC_DMA_GLOBAL_IER - MAC Interrupt enable

```
0      Global interrupt enable (0=Disable, 1=Enable)
```

ATH:020028h - MAC_DMA_TIMT_0 - Transmit Interrupt Mitigation Threshold
ATH:02002Ch - MAC_DMA_RIMT - Receive Interrupt Mitigation Threshold

```
0-15   Last packet threshold (LAST_PKT_THRESH)
16-31  First packet threshold (FIRST_PKT_THRESH)
```

ATH:020030h - MAC_DMA_TXCFG - MAC Transmit DMA size config register

In hw4, most bits are left undefined, however, hw4 DOES refer to the register as "MAC tx DMA size config", so one may assume that at least the SIZE value in bit0-2 does exist in hw4, too.

```
0-2    hw6: DMA_SIZE (maybe as in RXCFG below?)       ;-hw6 only (???)
3      -
4-9    Frame trigger level (TRIGLVL)
10     hw6: JUMBO_EN                                  ;-hw6 only (??)
11     ADHOC_BEACON_ATIM_TX_POLICY (hw6name: BCN_PAST_ATIM_DIS)
12     hw6: ATIM_DEFER_DIS                            ;\
13     -                                              ;
14     hw6: RTCI_DIS                                  ; hw6 only (?)
15-16  -                                              ;
17     hw6: DIS_RETRY_UNDERRUN                        ;
18     hw6: DIS_CW_INC_QUIET_COLL                     ;
19     hw6: RTS_FAIL_EXCESSIVE_RETRIES                ;/
```

Blurb...

```
MAC_DMA_FTRIG_IMMED = 0x00000000 ;bytes in PCU TX FIFO before air
MAC_DMA_FTRIG_64B   = 0x00000010 ;default
MAC_DMA_FTRIG_128B  = 0x00000020
MAC_DMA_FTRIG_192B  = 0x00000030
MAC_DMA_FTRIG_256B  = 0x00000040 ;5 bits total
```

ATH:020034h - MAC_DMA_RXCFG - MAC rx DMA size config register

```
0-2    DMA Size (0..7 = 4,8,16,32,64,128,256,512 bytes)
3-4    hw6: ZERO_LEN_DMA_EN                           ;-hw6: two bits?
4      hw4: Enable DMA of zero-length frame           ;-hw4: one bit?
5      hw6: JUMBO_EN                                  ;\
6      hw6: JUMBO_WRAP_EN                             ; hw6 only (?)
7      hw6: SLEEP_RX_PEND_EN                          ;/
```

Blurb...

```
MAC_DMA_RXCFG_DMASIZE_4B   = 0x00000000 ;DMA size 4 bytes (TXCFG + RXCFG)
MAC_DMA_RXCFG_DMASIZE_8B   = 0x00000001 ;DMA size 8 bytes
MAC_DMA_RXCFG_DMASIZE_16B  = 0x00000002 ;DMA size 16 bytes
MAC_DMA_RXCFG_DMASIZE_32B  = 0x00000003 ;DMA size 32 bytes
MAC_DMA_RXCFG_DMASIZE_64B  = 0x00000004 ;DMA size 64 bytes
MAC_DMA_RXCFG_DMASIZE_128B = 0x00000005 ;DMA size 128 bytes
MAC_DMA_RXCFG_DMASIZE_256B = 0x00000006 ;DMA size 256 bytes
MAC_DMA_RXCFG_DMASIZE_512B = 0x00000007 ;DMA size 512 bytes
```

ATH:020038h - MAC_DMA_RXJLA (R) ;hw6 only

```
31-2   DATA                                   (R) ;-hw6 only
```

ATH:020040h - MAC_DMA_MIBC - MAC MIB control register

```
0     counter overflow warning (WARNING)      (R)
1     freeze MIB counters (FREEZE)
2     clear MIB counters (CLEAR)
3     MIB counter strobe, increment all (STROBE)  (R)
```

ATH:020044h - MAC_DMA_TOPS - MAC timeout prescale count

```
  0-15  Timeout prescale (TIMEOUT)
```

ATH:020048h - MAC_DMA_RXNPTO - MAC no frame received timeout

```
  0-9   No frame received timeout (TIMEOUT)
```

ATH:02004Ch - MAC_DMA_TXNPTO - MAC no frame trasmitted timeout

```
  0-9   No frame transmitted timeout (TIMEOUT)
  10-19 QCU Mask (QCU 0-9)  ;QCU's for which frame completions will cause
          a reset of the no frame xmit'd timeout
```

ATH:020050h - MAC_DMA_RPGTO - MAC receive frame gap timeout

```
  0-9   Receive frame gap timeout (TIMEOUT)
```

ATH:020054h - MAC_DMA_RPCNT - MAC receive frame count limit ;hw4 only

```
  0-4   Receive frame count limit                       ;-hw4 only
```

ATH:020058h - MAC_DMA_MACMISC - MAC miscellaneous control/status register

```
  4      hw6: FORCE_PCI_EXT                             ;-hw6 only
  5-8    DMA observation bus mux select (DMA_OBS_MUXSEL)
  9-11   MISC observation bus mux select (MISC_OBS_MUXSEL)
  12-14  MAC observation bus mux select (lsb) (MISC_F2_OBS_LOW_MUXSEL)
  15-17  MAC observation bus mux select (msb) (MISC_F2_OBS_HIGH_MUXSEL)

 ____________ below in hw6 only ____________
```

ATH:02005Ch - MAC_DMA_INTER ;hw6 only

```
  0      REQ
  1-2    MSI_RX_SRC
  3-4    MSI_TX_SRC
```

ATH:020060h - MAC_DMA_DATABUF ;hw6 only

```
  0-11   LEN
```

ATH:020064h - MAC_DMA_GTT ;hw6 only
ATH:02006Ch - MAC_DMA_CST ;hw6 only

```
  0-15   COUNT
  16-31  LIMIT
```

ATH:020068h - MAC_DMA_GTTM ;hw6 only

```
  0      USEC_STROBE
  1      IGNORE_CHAN_IDLE
  2      RESET_ON_CHAN_IDLE
  3      CST_USEC_STROBE
  4      DISABLE_QCU_FR_ACTIVE_GTT
  5      DISABLE_QCU_FR_ACTIVE_BT
```

ATH:020070h - MAC_DMA_RXDP_SIZE ;hw6 only

```
  0-7    LP                              (R)
  8-12   HP                              (R)
```

ATH:020074h - MAC_DMA_RX_QUEUE_HP_RXDP ;hw6 only
ATH:020078h - MAC_DMA_RX_QUEUE_LP_RXDP ;hw6 only

```
  0-31   ADDR
```

ATH:0200E0h - MAC_DMA_DMADBG_0 (R) - hw6 only
ATH:0200E4h - MAC_DMA_DMADBG_1 (R) - hw6 only
ATH:0200E8h - MAC_DMA_DMADBG_2 (R) - hw6 only
ATH:0200ECh - MAC_DMA_DMADBG_3 (R) - hw6 only
ATH:0200F0h - MAC_DMA_DMADBG_4 (R) - hw6 only

ATH:0200F4h - MAC_DMA_DMADBG_5 (R) - hw6 only
ATH:0200F8h - MAC_DMA_DMADBG_6 (R) - hw6 only
ATH:0200FCh - MAC_DMA_DMADBG_7 (R) - hw6 only

```
  0-31   DATA                                    (R)  ;-hw6 only
```

ATH:020100h - MAC_DMA_QCU_TXDP_REMAINING_QCU_7_0 (R) ;hw6 only
ATH:020104h - MAC_DMA_QCU_TXDP_REMAINING_QCU_9_8 (R) ;hw6 only

```
  0-39   For QCU 0-9 (4bits each)                (R)  ;\hw6 only
  40-63  -                                            ;/
```

(see above) - MAC_DMA_TIMT_0 - hw4/hw6
ATH:020108h - MAC_DMA_TIMT_1 - hw6 only
ATH:02010Ch - MAC_DMA_TIMT_2 - hw6 only
ATH:020110h - MAC_DMA_TIMT_3 - hw6 only
ATH:020114h - MAC_DMA_TIMT_4 - hw6 only
ATH:020118h - MAC_DMA_TIMT_5 - hw6 only
ATH:02011Ch - MAC_DMA_TIMT_6 - hw6 only
ATH:020120h - MAC_DMA_TIMT_7 - hw6 only
ATH:020124h - MAC_DMA_TIMT_8 - hw6 only
ATH:020128h - MAC_DMA_TIMT_9 - hw6 only

```
  0-15   LAST_PKT_THRESH
  16-31  FIRST_PKT_THRESH
```

ATH:02012Ch - MAC_DMA_CHKACC - hw6 only

```
  0      CHKSUM_SEL
```

DSi Atheros Wifi - Internal I/O - 020080h - WMAC IRQ Interrupt (hw4/hw6)

ATH:020080h - MAC_DMA_ISR(_P) - MAC Primary interrupt status register
ATH:0200A0h - MAC_DMA_IMR(_P) - MAC Primary interrupt mask register
ATH:0200C0h - MAC_DMA_ISR(_P)_RAC - MAC Primary interrupt read-and-clear

Interrupt Status Registers

Only the bits in the ISR_P register and the IMR_P registers control whether the MAC's INTA# output is asserted. The bits in the secondary interrupt status/mask registers control what bits are set in the primary interrupt status register; however the IMR_S* registers DO NOT determine whether INTA# is asserted. That is INTA# is asserted only when the logical AND of ISR_P and IMR_P is non-zero. The secondary interrupt mask/status registers affect what bits are set in ISR_P but they do not directly affect whether INTA# is asserted.

Interrupt Mask Registers

Only the bits in the IMR control whether the MAC's INTA# output will be asserted. The bits in the secondary interrupt mask registers control what bits get set in the primary interrupt status register; however the IMR_S* registers DO NOT determine whether INTA# is asserted.

Read-and-Clear Registers

The read-and-clear registers are read-only (R) "shadow copies" of the interrupt status registers, with read-and-clear access.

```
  0     At least one frame received sans errors  ;\
  1     Receive interrupt request                ;
  2     Receive error interrupt                  ; RX
  3     No frame received within timeout clock   ;
  4     Received descriptor empty interrupt      ;
  5     Receive FIFO overrun interrupt           ;/
  6     Transmit okay interrupt                  ;\       ;<-- ISR_S0.Bit0..9
  7     Transmit interrupt request               ;
  8     Transmit error interrupt                 ; TX     ;<-- ISR_S1.Bit0..9
  9     No frame transmitted interrupt           ;
```

```
10    Transmit descriptor empty interrupt      ;
11    Transmit FIFO underrun interrupt         ;/       ;<-- ISR_S2.Bit0..9
12    MIB interrupt - see MIBC
13    Software interrupt
14    PHY receive error interrupt
15    Key-cache miss interrupt
16    Beacon rssi high threshold interrupt  ;aka Beacon rssi hi threshold
17    Beacon threshold interrupt            ;aka Beacon rssi lo threshold
18    Beacon missed interrupt
19    Maximum transmit interrupt rate
20    Beacon not ready interrupt            ;aka BNR interrupt
21    An unexpected bus error has occurred
22    -
23    Beacon Misc (TIM, CABEND, DTIMSYNC, BCNTO)        ;<-- ISR_S2.Bit24..27
24    Maximum receive interrupt rate
25    QCU CBR overflow interrupt                        ;<-- ISR_S3.Bit0..9
26    QCU CBR underrun interrupt                        ;<-- ISR_S3.Bit16..27
27    QCU scheduling trigger interrupt                  ;<-- ISR_S4.Bit0..9
28    GENTMR interrupt (aka GENERIC_TIMERS... and/or ISR_S5?)
29    HCFTO interrupt
30    Transmit completion mitigation interrupt
31    Receive completion mitigation interrupt
```

ATH:020084h - MAC_DMA_ISR_S0 - MAC Secondary Interrupt 0 Stat TX OK/DESC
ATH:0200A4h - MAC_DMA_IMR_S0 - MAC Secondary Interrupt 0 Mask TX OK/DESC
ATH:0200C4h - MAC_DMA_ISR_S0_S - MAC Secondary Interrupt 0 Read-and-Clear

```
0-9   TXOK (QCU 0-9)                                   ;--> Primary_ISR.Bit6
16-27 TXDESC (QCU 0-9)                                 ;--> Primary_ISR. ??
```

ATH:020088h - MAC_DMA_ISR_S1 - MAC Secondary Interrupt 1 Stat TX ERR/EOL
ATH:0200A8h - MAC_DMA_IMR_S1 - MAC Secondary Interrupt 1 Mask TX ERR/EOL
ATH:0200C8h - MAC_DMA_ISR_S1_S - MAC Secondary Interrupt 1 Read-and-Clear

```
0-9   TXERR (QCU 0-9)                                  ;--> Primary_ISR.Bit8
16-27 TXEOL (QCU 0-9)
```

ATH:02008Ch - MAC_DMA_ISR_S2 - MAC Secondary Interrupt 2 Stat TX URN/MISC
ATH:0200ACh - MAC_DMA_IMR_S2 - MAC Secondary Interrupt 2 Mask TX URN/MISC
ATH:0200CCh (hw6:000200D0h?) - MAC_DMA_ISR_S2_S - MAC .. 2 Read-and-Clear

```
0-9   TXURN (QCU 0-9)                                  ;--> Primary_ISR.Bit11
10    -
11    RX_INT                  ;RX
12    WL_STOMPED
13    RX_PTR_BAD              ;RX
14    BT_LOW_PRIORITY_RISING
15    BT_LOW_PRIORITY_FALLING
16    BB_PANIC_IRQ
17    BT_STOMPED
18    BT_ACTIVE_RISING
19    BT_ACTIVE_FALLING
20    BT_PRIORITY_RISING
21    BT_PRIORITY_FALLING
22    CST
23    GTT
24    TIM                               ;\
25    CABEND                            ; Beacon Misc --> Primary_ISR.Bit23
26    DTIMSYNC                          ;
27    BCNTO                             ;/
28    CABTO
29    DTIM
30    TSFOOR
31    -
```

ATH:020090h - MAC_DMA_ISR_S3 - MAC Secondary Interrupt 3 Stat QCBR OVF/URN
ATH:0200B0h - MAC_DMA_IMR_S3 - MAC Secondary Interrupt 3 Mask QCBR OVF/URN
ATH:0200D0h (hw6:000200D4h?) - MAC_DMA_ISR_S3_S - MAC .. 3 Read-and-Clear

```
  0-9   QCBROVF (QCU 0-9)                                ;--> Primary_ISR.Bit25
  16-27 QCBRURN (QCU 0-9)                                ;--> Primary_ISR.Bit26
```

ATH:020094h - MAC_DMA_ISR_S4 - MAC Secondary Interrupt 4 Stat QTRIG
ATH:0200B4h - MAC_DMA_IMR_S4 - MAC Secondary Interrupt 4 Mask QTRIG
ATH:0200D4h (hw6:000200D8h?) - MAC_DMA_ISR_S4_S - MAC .. 4 Read-and-Clear

```
  0-9   QTRIG (QCU 0-9)                                  ;--> Primary_ISR.Bit27
```

ATH:020098h - MAC_DMA_ISR_S5 - MAC Secondary Interrupt 5 Stat TIMERS
ATH:0200B8h - MAC_DMA_IMR_S5 - MAC Secondary Interrupt 5 Mask TIMERS
ATH:0200D8h (hw6:000200DCh?) - MAC_DMA_ISR_S5_S - MAC .. 5 Read-and-Clear

```
  0     TBTT_TIMER_TRIGGER              ;-TBTT timer
  1     DBA_TIMER_TRIGGER               ;\
  2     SBA_TIMER_TRIGGER               ;
  3     HCF_TIMER_TRIGGER               ;
  4     TIM_TIMER_TRIGGER               ; timer's
  5     DTIM_TIMER_TRIGGER              ;
  6     QUIET_TIMER_TRIGGER             ;
  7     NDP_TIMER_TRIGGER               ;
  8-15  GENERIC_TIMER2_TRIGGER          ;/
  16    TIMER_OVERFLOW                  ;<-- which timer overflow ?
  17    DBA_TIMER_THRESHOLD             ;\
  18    SBA_TIMER_THRESHOLD             ;
  19    HCF_TIMER_THRESHOLD             ;
  20    TIM_TIMER_THRESHOLD             ; threshold's
  21    DTIM_TIMER_THRESHOLD            ;
  22    QUIET_TIMER_THRESHOLD           ;
  23    NDP_TIMER_THRESHOLD             ;
  24-31 GENERIC_TIMER2_THRESHOLD        ;/
```

ATH:02009Ch - MAC_DMA_ISR_S6 - hw6 only ;Interrupt 6 Stat UNKNOWN(?)
ATH:0200BCh - MAC_DMA_IMR_S6 - hw6 only ;Interrupt 6 Mask UNKNOWN(?)
ATH:0200CCh (hw6:really?) - MAC_DMA_ISR_S6_S (R) shadow - hw6 only

```
  0-31   ?? (probably related to the new "hw6" registers in MAC DMA chapter)
```

Note: According to hw6.0 source code, "S6_S" has been accidently inserted between "S1_S" and "S2_S" (thus smashing the old hw4-style port numbering for "S2_S thru S5_S") (not checked if that's a source code bug, or if it's actually implemented as so in real hardware).
The hw6.0 source code leaves ALL primary and secondary IRQ bits undocumented, thw above descriptions are based on hw4 (but hw6 might have some added/changed bits here and there).

DSi Atheros Wifi - Internal I/O - 020800h - WMAC QCU Queue (hw4/hw6)

ATH:020800h..020824h - MAC_QCU_TXDP[0..9] aka MAC_DMA_Q(0..9)_TXDP

```
  0-31   DATA  ...   unspecified     ;MAC Transmit Queue descriptor pointer
```

ATH:020840h - MAC_QCU_TXE aka MAC_DMA_Q_TXE - MAC Transmit Queue enable (R)
ATH:020880h - MAC_QCU_TXD aka MAC_DMA_Q_TXD - MAC Transmit Queue disable

```
  0-9    DATA
```

ATH:0208C0h..0208E4h - MAC_QCU_CBR[0..9] aka MAC_DMA_Q(0..9)_CBRCFG

```
  0-23  CBR interval (us)      (INTERVAL)       ;\MAC CBR configuration
```

```
  24-31 CBR overflow threshold (OVF_THRESH)     ;/
```

ATH:020900h..020924h - MAC_QCU_RDYTIME[0..9] aka MAC_DMA_Q(0..9)_RDYTIMECFG
```
  0-23  CBR interval (us)      (DURATION)       ;\MAC ReadyTime configuration
  24    CBR enable             (EN)             ;/
```

ATH:020940h - MAC_QCU_ONESHOT_ARM_SC aka MAC_DMA_Q_ONESHOTMAC_DMAM_SC - Set
ATH:020980h - MAC_QCU_ONESHOT_ARM_CC aka MAC_DMA_Q_ONESHOTMAC_DMAM_CC - Clr
MAC OneShotArm set/clear control
```
  0-9    SET/CLEAR
```

ATH:0209C0h..0209E4h - MAC_QCU_MISC[0..9] aka MAC_DMA_Q(0..9)_MISC
MAC Miscellaneous QCU settings
```
  0-3   Frame Scheduling Policy mask (FSP):
          0=ASAP                    ;\
          1=CBR                     ; defined as so for
          2=DMA Beacon Alert gated  ; hw4 (maybe same
          3=TIM gated               ; for hw6)
          4=Beacon-sent-gated       ;/
  4     OneShot enable (ONESHOT_EN)
  5     CBR expired counter disable incr (NOFR, empty q)
  6     CBR expired counter disable incr (NOBCNFR, empty beacon q)
  7     Beacon use indication (IS_BCN)
  8     CBR expired counter limit enable (CBR_EXP_INC_LIMIT)
  9     Enable TXE cleared on ReadyTime expired or VEOL (TXE_CLR_ON_CBR_END)
  10    CBR expired counter reset (MMR_CBR_EXP_CNT_CLR_EN)
  11    FR_ABORT_REQ_EN         DCU frame early termination request control
```

ATH:020A00h..020A24h - MAC_QCU_CNT[0..9] aka MAC_DMA_Q(0..9)_STS
```
  0-1    FR_PEND: Pending Frame Count (R)        ;\MAC Misc QCU status/counter
  8-15   CBR_EXP: CBR expired counter (R)        ;/
```

ATH:020A40h - MAC_QCU_RDYTIME_SHDN aka MAC_DMA_Q_RDYTIMESHDN
```
  0-9    SHUTDOWN: MAC ReadyTimeShutdown status (flags for QCU 0-9 ?)
```

```
 _____________ below in hw6 only _____________
```

ATH:020830h - MAC_QCU_STATUS_RING_START ;hw6 only
```
  0-31   ADDR
```

ATH:020834h - MAC_QCU_STATUS_RING_END ;hw6 only
```
  0-31   ADDR
```

ATH:020838h - MAC_QCU_STATUS_RING_CURRENT (R) ;hw6 only
```
  0-31   ADDRESS                        (R)
```

ATH:020A44h - MAC_QCU_DESC_CRC_CHK ;hw6 only
```
  0      EN
```

ATH:020A48h - MAC_QCU_EOL ;hw6 only
```
  0-9    DUR_CAL_EN
```

DSi Atheros Wifi - Internal I/O - 021000h - WMAC DCU (hw4/hw6)

ATH:021000h..021024h - MAC_DCU_QCUMASK[0..9] aka MAC_DMA_D(0..9)_QCUMASK

The "DCU_QCU" Mask is some two-dimensional array: An array of ten DCU registers, each containing an array of ten QCU bits.

```
0-9   QCU Mask (QCU 0-9)
```

ATH:021030h - MAC_DCU_GBL_IFS_SIFS aka MAC_DMA_D_GBL_IFS_SIFS

```
0-15   DURATION                      ;-DCU global SIFS settings
```

ATH:021040h..021064h - MAC_DCU_LCL_IFS[0..9] aka MAC_DMA_D(0..9)_LCL_IFS

```
0-9    CW_MIN                              ;\
10-19  CW_MAX                              ; MAC DCU-specific IFS settings
20-27  AIFS                                ;
28     hw6: LONG_AIFS     ;-hw6 only       ;
29-31  -                                   ;/
```

Note: Even though this field is 8 bits wide the maximum supported AIFS value is FCh. Setting the AIFS value to FDh,FEh,FFh will not work correctly and will cause the DCU to hang (said to be so; at least for hw4).

ATH:021070h - MAC_DCU_GBL_IFS_SLOT aka MAC_DMA_D_GBL_IFS_SLOT

```
0-15   DURATION       ;DC global slot interval
```

ATH:021080h..0210A4h - MAC_DCU_RETRY_LIMIT[0..9] aka MAC_DMA_D(0..9)_RETR..

```
0-3   frame RTS failure limit (FRFL)         ;\
4-7   -                                      ; MAC Retry limits
8-13  station RTS failure limit (SRFL)       ;
14-19 station short retry limit (SDFL)       ;
20-31 -                                      ;/
```

ATH:0210B0h - MAC_DCU_GBL_IFS_EIFS aka MAC_DMA_D_GBL_IFS_EIFS

```
0-15   DURATION                              ;-DCU global EIFS setting
```

ATH:0210C0h..0210E4h - MAC_DCU_CHANNEL_TIME[0..9] aka MAC_DMA_D(0..9)_CH..

```
0-15  ChannelTime duration (us) (DURATION)   ;\MAC ChannelTime settings
16    ChannelTime enable        (ENABLE)     ;/
```

ATH:0210F0h - MAC_DCU_GBL_IFS_MISC aka MAC_DMA_D_GBL_IFS_MISC

```
0-2    LFSR slice select (LFSR_SLICE_SEL)       ;\
3      Turbo mode indication (TURBO_MODE)       ;
4-9    hw6: SIFS_DUR_USEC           ;-hw6 only  ;
10-19  -                                        ;
20-21  DCU arbiter delay (ARB_DLY)              ; DCU global misc.
22     hw6: SIFS_RST_UNCOND         ;\          ; IFS settings
23     hw6: AIFS_RST_UNCOND         ;           ;
24     hw6: LFSR_SLICE_RANDOM_DIS   ; hw6 only  ;
25-26  hw6: CHAN_SLOT_WIN_DUR       ;           ;
27     hw6: CHAN_SLOT_ALWAYS        ;/          ;
28     IGNORE_BACKOFF                          ;
29     hw6: SLOT_COUNT_RST_UNCOND   ;-hw6 only  ;
30-31  -                                        ;/
```

ATH:021100h..021124h - MAC_DCU_MISC[0..9] aka MAC_DMA_D(0..9)_MISC

```
0-5    BKOFF_THRESH                  ;\
6      SFC_RST_AT_TS_END_EN          ;
7      CW_RST_AT_TS_END_DIS          ;
8      FRAG_BURST_WAIT_QCU_EN        ;
9      FRAG_BURST_BKOFF_EN           ; MAC Miscellaneous
10     -                             ; DCU-specific settings
11     HCF_POLL_EN                   ;
12     BKOFF_PF                      ; (specified as so for hw6)
13     -                             ; (hw4 bit numbers are undocumented,
14-15  VIRT_COLL_POLICY              ; although... actually the SAME bits
16     IS_BCN                        ; ARE documented, but for the "EOL"
17     ARB_LOCKOUT_IF_EN             ; registers instead of for "MISC"...?)
```

```
  18     LOCKOUT_GBL_EN                ;
  19     LOCKOUT_IGNORE                ;
  20     SEQNUM_FREEZE                 ;
  21     POST_BKOFF_SKIP               ;
  22     VIRT_COLL_CW_INC_EN           ;
  23     RETRY_ON_BLOWN_IFS_EN         ;
  24     SIFS_BURST_CHAN_BUSY_IGNORE   ;
  25-31  -                             ;/
```

ATH:021140h - MAC_DCU_SEQ aka MAC_DMA_D_SEQNUM - MAC Frame sequence number

```
  0-31   NUM
```

BUG: The original hw4 header file used sorted addresses, but it's been having 00021140h placed between 000211A4h and 00021230h... either it's just mis-sorted, or the address is mis-spelled? Probably just mis-sorted, because hw6 does confirm the address.

ATH:021270h - MAC_DCU_PAUSE aka MAC_DMA_D_TXPSE

```
  0-9    REQUEST                     ;\DCU transmit pause control/status
  16     STATUS              (R)     ;/
```

```
 _____________ below in hw4 only _____________
```

ATH:021230h - MAC_DMA_D_FPCTL - DCU frame prefetch settings - hw4 only?

```
  ...    unspecified
```

ATH:021180h..021xx4h - MAC_DMA_D(0..9)_EOL - hw4 only (removed in hw6)

;BUG: entry D(8..9) are officially at 21200h..21204h ;\which is implemented
;BUG: entry D(8..9) are *intended* at 211A0h..211A4h ;/in actual hardware?
;BUG: below bits are probably meant to be "MISC" (not "EOL") ?

```
  0-5    Backoff threshold
  6      End of transmission series station RTS/data failure count reset policy
  7      End of transmission series CW reset policy
  8      Fragment Starvation Policy
  9      Backoff during a frag burst
  10     -
  11     HFC poll enable
  12     Backoff persistence factor setting
  13
  14-15  Mask for Virtual collision handling policy
           (0=Normal, 1=Ignore, 2..3=Unspecified)
  16     Beacon use indication
  17-18  Mask for DCU arbiter lockout control
           (0=No Lockout, 1=Intra-frame, 2=Global, 3=Unspecified)
  19     DCU arbiter lockout ignore control
  20     Sequence number increment disable
  21     Post-frame backoff disable
  22     Virtual coll. handling policy
  23     Blown IFS handling policy
  24-31  -
```

BUG: The official source code assigns incorrect "BCD-style" values to MAC_DMA_D8_EOL and MAC_DMA_D9_EOL, the source code does also have a MAC_DMA_DEOL_ADDRESS(i) macro function; that function would work okay even with i=8..9.

```
 _____________ below in hw6 only _____________
```

ATH:0212B0h - MAC_DCU_WOW_KACFG - hw6 only

```
  0      TX_EN
  1      TIM_EN
  4-11   BCN_CNT
  12-23  RX_TIMEOUT_CNT
```

ATH:0212F0h - MAC_DCU_TXSLOT - hw6 only

0-15	MASK	

ATH:02143Ch - MAC_DCU_TXFILTER_CLEAR - hw6 only

0-31	DATA	

ATH:02147Ch - MAC_DCU_TXFILTER_SET - hw6 only

0-31	DATA	

ATH:021038h - MAC_DCU_TXFILTER_DCU0_31_0 - hw6 only
ATH:021078h - MAC_DCU_TXFILTER_DCU0_63_32 - hw6 only
ATH:0210B8h - MAC_DCU_TXFILTER_DCU0_95_64 - hw6 only
ATH:0210F8h - MAC_DCU_TXFILTER_DCU0_127_96 - hw6 only

0-31	DATA	

ATH:02103Ch - MAC_DCU_TXFILTER_DCU8_31_0 - hw6 only (R)
ATH:02107Ch - MAC_DCU_TXFILTER_DCU8_63_32 - hw6 only (R)
ATH:0210BCh - MAC_DCU_TXFILTER_DCU8_95_64 - hw6 only (R)
ATH:0210FCh - MAC_DCU_TXFILTER_DCU8_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:021138h - MAC_DCU_TXFILTER_DCU1_31_0 - hw6 only (R)
ATH:021178h - MAC_DCU_TXFILTER_DCU1_63_32 - hw6 only (R)
ATH:0211B8h - MAC_DCU_TXFILTER_DCU1_95_64 - hw6 only (R)
ATH:0211F8h - MAC_DCU_TXFILTER_DCU1_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:02113Ch - MAC_DCU_TXFILTER_DCU9_31_0 - hw6 only (R)
ATH:02117Ch - MAC_DCU_TXFILTER_DCU9_63_32 - hw6 only (R)
ATH:0211BCh - MAC_DCU_TXFILTER_DCU9_95_64 - hw6 only (R)
ATH:0211FCh - MAC_DCU_TXFILTER_DCU9_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:021238h - MAC_DCU_TXFILTER_DCU2_31_0 - hw6 only (R)
ATH:021278h - MAC_DCU_TXFILTER_DCU2_63_32 - hw6 only (R)
ATH:0212B8h - MAC_DCU_TXFILTER_DCU2_95_64 - hw6 only (R)
ATH:0212F8h - MAC_DCU_TXFILTER_DCU2_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:021338h - MAC_DCU_TXFILTER_DCU3_31_0 - hw6 only (R)
ATH:021378h - MAC_DCU_TXFILTER_DCU3_63_32 - hw6 only (R)
ATH:0213B8h - MAC_DCU_TXFILTER_DCU3_95_64 - hw6 only (R)
ATH:0213F8h - MAC_DCU_TXFILTER_DCU3_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:021438h - MAC_DCU_TXFILTER_DCU4_31_0 - hw6 only (R)
ATH:021478h - MAC_DCU_TXFILTER_DCU4_63_32 - hw6 only (R)
ATH:0214B8h - MAC_DCU_TXFILTER_DCU4_95_64 - hw6 only (R)
ATH:0214F8h - MAC_DCU_TXFILTER_DCU4_127_96 - hw6 only (R)

0-31	DATA	(R)

ATH:021538h - MAC_DCU_TXFILTER_DCU5_31_0 - hw6 only (R)
ATH:021578h - MAC_DCU_TXFILTER_DCU5_63_32 - hw6 only (R)
ATH:0215B8h - MAC_DCU_TXFILTER_DCU5_95_64 - hw6 only (R)
ATH:0215F8h - MAC_DCU_TXFILTER_DCU5_127_96 - hw6 only (R)

```
  0-31   DATA                          (R)
```

ATH:021638h - MAC_DCU_TXFILTER_DCU6_31_0 - hw6 only (R)
ATH:021678h - MAC_DCU_TXFILTER_DCU6_63_32 - hw6 only (R)
ATH:0216B8h - MAC_DCU_TXFILTER_DCU6_95_64 - hw6 only (R)
ATH:0216F8h - MAC_DCU_TXFILTER_DCU6_127_96 - hw6 only (R)
```
  0-31   DATA                          (R)
```

ATH:021738h - MAC_DCU_TXFILTER_DCU7_31_0 - hw6 only (R)
ATH:021778h - MAC_DCU_TXFILTER_DCU7_63_32 - hw6 only (R)
ATH:0217B8h - MAC_DCU_TXFILTER_DCU7_95_64 - hw6 only (R)
ATH:0217F8h - MAC_DCU_TXFILTER_DCU7_127_96 - hw6 only (R)
```
  0-31   DATA                          (R)
```

DSi Atheros Wifi - Internal I/O - 028000h - WMAC PCU (hw2/hw4/hw6)

ATH:028000h - MAC_PCU_STA_ADDR_L32 (aka hw2: REG_STA_ID0) ;hw2/hw4/hw6
ATH:028004h - MAC_PCU_STA_ADDR_U16 (aka hw2: REG_STA_ID1) ;hw2/hw4/hw6
```
  0-47   ADDR (called STA_ADDR in hw2) (local MAC address)
  48     STA_AP (called AP in hw2)
  49     ADHOC
  50     PW_SAVE (called PWR_SV in hw2)
  51     KEYSRCH_DIS (called NO_KEYSRCH in hw2)
  52     PCF
  53     USE_DEFANT (called USE_DEF_ANT in hw2)
  54     DEFANT_UPDATE (called DEF_ANT_UPDATE in hw2)
  55     RTS_USE_DEF (called RTS_DEF_ANT in hw2)
  56     ACKCTS_6MB
  57     BASE_RATE_11B (called RATE_11B in hw2)
  58     SECTOR_SELF_GEN
  59     CRPT_MIC_ENABLE
  60     KSRCH_MODE
  61     PRESERVE_SEQNUM
  62     CBCIV_ENDIAN
  63     ADHOC_MCAST_SEARCH
```

ATH:028008h - MAC_PCU_BSSID_L32 (aka hw2: REG_BSS_ID0) ;hw2/hw4/hw6
ATH:02800Ch - MAC_PCU_BSSID_U16 (aka hw2: REG_BSS_ID1) ;hw2/hw4/hw6
```
  0-47   ADDR (called BSSID in hw2)
  48-58  AID (11bit, although claimed to be 16bit wide, bit48-63 in hw2)
```

ATH:0282B0h/02839Ch/028398h - MAC_PCU_BSSID2_L32 ;hw2/hw4/hw6
ATH:0282B4h/0283A0h/02839Ch - MAC_PCU_BSSID2_U16 ;hw2/hw4/hw6
```
  0-47   ADDR    (hw2: SEC_BSSID, ini:0)               ;\hw2/hw4/hw6
  48     ENABLE  (hw2: SEC_BSSID_ENABLE, ini:0)        ;/
  49-51  -
  52-62  hw6: AID                                      ;-hw 6 only
  63     -
```
Observe that hw2/hw6 port addresses are different here (unlike as usually).

ATH:028010h - MAC_PCU_BCN_RSSI_AVE (aka hw2:BCNRSSI) (R) ;hw2/hw4/hw6
```
  0-11   AVE_VALUE (aka hw2:BCN_RSSI_AVE ini:800h) (R)  ;-hw2/hw4/hw6
  16-27  hw6: AVE_VALUE2                           (R)  ;-hw6 only
```

ATH:028014h - MAC_PCU_ACK_CTS_TIMEOUT (aka hw2:REG_TIME_OUT) ;hw2/hw4/hw6
```
  0-13   ACK_TIMEOUT (aka 16bit wide, bit0-15: ACK_TIME_OUT in hw2)
```

```
  16-29  CTS_TIMEOUT (aka 16bit wide, bit16-31: CTS_TIME_OUT in hw2)
```

ATH:028018h - MAC_PCU_BCN_RSSI_CTL (aka hw2:BCNSIG) ;hw2/hw4/hw6

```
  0-7    RSSI_LOW_THRESH  (aka hw2: BCN_RSSI_LO_THR, ini:0)
  8-15   MISS_THRESH      (aka hw2: BCN_MISS_THR, ini:FFh)
  16-23  RSSI_HIGH_THRESH (aka hw2: BCN_RSSI_HI_THR, ini:7Fh)
  24-28  WEIGHT           (aka hw2: BCN_RSSI_WEIGHT, ini:0)
  29     RESET            (aka hw2: BCN_RSSI_RESET)
```

ATH:02801Ch - MAC_PCU_USEC_LATENCY (aka hw2:REG_USEC) ;hw2/hw4/hw6

```
  <-- hw2 (REG_USEC) -->    <--- hw4/hw6 ---------->
  0-6    USEC     (7bit)    0-7    USEC       (8bit)
  7-13   USEC32   (7bit)    8-13   -          (-)
  14-18  TX_DELAY (5bit)    14-22  TX_LATENCY (9bit)
  19-24  RX_DELAY (6bit)    23-28  RX_LATENCY (6bit)
```

ATH:02803Ch/028024h/02803Ch - MAC_PCU_RX_FILTER ;hw2/hw4/hw6

```
  0      UNICAST
  1      MULTICAST
  2      BROADCAST
  3      CONTROL
  4      BEACON
  5      PROMISCUOUS
  6      XR_POLL
  7      PROBE_REQ
  8      hw2: outcommented: SYNC        ;\hw4 and hw6 (outcommented in hw2)
  8      hw4/hw6: SYNC_FRAME            ;/
  9      MY_BEACON
  10     hw4/hw6: COMPRESSED_BAR        ;\
  11     hw4/hw6: COMPRESSED_BA         ;
  12     hw4/hw6: UNCOMPRESSED_BA_BAR   ; hw4 and hw6
  13     hw4/hw6: ASSUME_RADAR          ;
  14     hw4/hw6: PS_POLL               ;
  15     hw4/hw6: MCAST_BCAST_ALL       ;
  16     hw4/hw6: RST_DLMTR_CNT_DISABLE ;/
  17     hw4: FROM_TO_DS                ;\
  18-23  hw4: GENERIC_FTYPE             ; hw4 only (moved to bit20-28 in hw6)
  24-25  hw4: GENERIC_FILTER            ;/
  17     hw6: HW_BCN_PROC_ENABLE        ;\
  18     hw6: MGMT_ACTION_MCAST         ; hw6 only
  19     hw6: CONTROL_WRAPPER           ;
  20     hw6: FROM_TO_DS                ;   ;\these bits were formerly
  21-26  hw6: GENERIC_FTYPE             ;   ; in bit17-25 in hw4)
  27-28  hw6: GENERIC_FILTER            ;   ;/
  29     hw6: MY_BEACON2                ;/
```

ATH:028040h/028028h/028040h - MAC_PCU_MCAST_FILTER_L32 ;hw2/hw4/hw6
ATH:028044h/02802Ch/028044h - MAC_PCU_MCAST_FILTER_U32 ;hw2/hw4/hw6

```
  0-63   VALUE (aka hw2: unspecified)
```

ATH:028048h/028030h/028048h - MAC_PCU_DIAG_SW ;hw2/hw4/hw6

```
  0      INVALID_KEY_NO_ACK        (aka hw2:DIS_WEP_ACK          ;ini:0)
  1      NO_ACK                    (aka hw2:DIS_ACK              ;ini:0)
  2      NO_CTS                    (aka hw2:DIS_CTS              ;ini:0)
  3      NO_ENCRYPT                (aka hw2:DIS_ENC              ;ini:0)
  4      NO_DECRYPT                (aka hw2:DIS_DEC              ;ini:0)
  5      HALT_RX                   (aka hw2:DIS_RX               ;ini:0)
  6      LOOP_BACK                 (aka hw2:LOOP_BACK            ;ini:0)
  7      CORRUPT_FCS               (aka hw2:CORR_FCS             ;ini:0)
  8      DUMP_CHAN_INFO            (aka hw2:CHAN_INFO            ;ini:0)
  9-16   - (aka hw2: RESERVED)     (aka hw2:RESERVED             ;ini:0)
  17     ACCEPT_NON_V0             (aka hw2:ACCEPT_NONV0         ;ini:0)
  18-19  OBS_SEL_1_0               (aka hw2:OBS_SEL_0_1          ;ini:0)
```

```
  20     RX_CLEAR_HIGH             (aka hw2:RXCLR_HIGH           ;ini:0)
  21     IGNORE_NAV                (aka hw2:IGNORE_NAV           ;ini:0)
  22     CHAN_IDLE_HIGH            (aka hw2:CHANIDLE_HIGH        ;ini:0)
  23     PHYERR_ENABLE_EIFS_CTL    (aka hw2:PHYERR_ENABLE_NEW    ;ini:0)
  24     DUAL_CHAIN_CHAN_INFO      (aka hw2:DUAL_CHAIN_CHAN_INFO ;ini:0)
  25     FORCE_RX_ABORT            (aka hw2:FORCE_RX_ABORT       ;ini:0)
  26     SATURATE_CYCLE_CNT        (aka hw2:SATURATE_CYCLE_CNT   ;ini:0)
  27     OBS_SEL_2                 (aka hw2:OBS_SEL_2            ;ini:0)
  28     hw4/hw6: RX_CLEAR_CTL_LOW             ;\
  29     hw4/hw6: RX_CLEAR_EXT_LOW             ; hw4/hw6 only
  30-31  hw4/hw6: DEBUG_MODE                   ;/
```

ATH:028054h/028034h/028054h - MAC_PCU_TST_ADDAC ;hw2/hw4/hw6

```
  0      hw2: TEST_MODE                ;\
  1      hw2: TEST_LOOP                ; hw2 (moved to bit1-14 in hw4/hw6)
  2-12   hw2: LOOP_LEN                 ;
  13     hw2: TEST_UPPER_8B            ;/
  14     hw2: TEST_MSB                 ;-hw2 only
  15     hw2: TEST_CAPTURE             ;-hw2 (moved to bit19 in hw4/hw6)
```

Differently arranged in hw4/hw6:

```
  0      hw4/hw6: CONT_TX              ;-hw4/hw6 only
  1      hw4/hw6: TESTMODE             ;\
  2      hw4/hw6: LOOP                 ; hw4/hw6 (formerly bit0-13 in hw2)
  3-13   hw4/hw6: LOOP_LEN             ;
  14     hw4/hw6: UPPER_8B             ;/
  15     hw6:     SAMPLE_SIZE_2K       ;-hw6 only
  16     hw4/hw6: TRIG_SEL             ;\
  17     hw4/hw6: TRIG_POLARITY        ; hw4/hw6 only
  18     hw4/hw6: CONT_TEST        (R) ;/
  19     hw4/hw6: TEST_CAPTURE         ;-hw4/hw6 (formerly bit15 in hw2)
  20     hw4/hw6: TEST_ARM             ;-hw4/hw6 only
```

ATH:028058h/028038h/028058h - MAC_PCU_DEF_ANTENNA ;hw2/hw4/hw6

```
  0-23   VALUE                         ;\hw4/hw6 (and maybe hw2, too)
  24     TX_DEF_ANT_SEL                ;/
  25     hw6: SLOW_TX_ANT_EN           ;\
  26     hw6: TX_CUR_ANT               ; hw6 only
  27     hw6: FAST_DEF_ANT             ;/
  28     RX_LNA_CONFIG_SEL             ;-hw4/hw6 (and maybe hw2, too)
  29     hw6: FAST_TX_ANT_EN           ;\
  30     hw6: RX_ANT_EN                ; hw6 only
  31     hw6: RX_ANT_DIV_ON            ;/
```

ATH:02805Ch/02803Ch/02805Ch - MAC_PCU_AES_MUTE_MASK_0 ;hw2/hw4/hw6

```
  0-15   FC      (hw2: ini:C7FFh)
  16-31  QOS     (hw2: ini:FFFFh)
```

ATH:028060h/028040h/028060h - MAC_PCU_AES_MUTE_MASK_1 ;hw2/hw4/hw6

```
  0-15   SEQ     (hw2: ini:000Fh)
  16-31  FC_MGMT (hw2: ini:E7FFh)
```

ATH:028064h/028044h/028064h - MAC_PCU_GATED_CLKS ;hw2/hw4/hw6

```
  0      -         (aka hw2: outcommented: SYNC, ini:1)
  1      GATED_TX  (aka hw2: TX, ini:0)
  2      GATED_RX  (aka hw2: RX, ini:0)
  3      GATED_REG (aka hw2: REG, ini:0)
```

ATH:028068h/028048h/028068h - MAC_PCU_OBS_BUS_2 (R) ;hw2/hw4/hw6

```
  0-17   VALUE (aka hw2: OBS_BUS)              (R)  ;-hw2/hw4/hw6
  18-21  hw6: WCF_STATE                        (R)  ;\
  22     hw6: WCF0_FULL                        (R)  ;
  23     hw6: WCF1_FULL                        (R)  ; hw6 only
```

```
  24-28  hw6: WCF_COUNT                          (R)  ;
  29     hw6: MACBB_ALL_AWAKE                    (R)  ;/
```

ATH:02806Ch/02804Ch/02806Ch - MAC_PCU_OBS_BUS_1 (R) ;hw2/hw4/hw6

```
  0      PCU_DIRECTED                            (R)
  1      PCU_RX_END                              (R)
  2      RX_WEP                                  (R)
  3      RX_MY_BEACON                            (R)
  4      FILTER_PASS                             (R)
  5      TX_HCF                                  (R)
  6      TM_QUIET_TIME    (aka hw2: QUIET_TIME)  (R)
  7      PCU_CHANNEL_IDLE (aka hw2: CHAN_IDLE)   (R)
  8      TX_HOLD                                 (R)
  9      TX_FRAME                                (R)
  10     RX_FRAME                                (R)
  11     RX_CLEAR                                (R)
  12-17  WEP_STATE                               (R)
  20-23  hw2: RX_STATE (4bit)                    (R)  ;\hw2 (less bits)
  24-28  hw2: TX_STATE (5bit)                    (R)  ;/
  20-24  hw4/hw6: RX_STATE (5bit)                (R)  ;\hw4/hw6 (one more
  25-30  hw4/hw6: TX_STATE (6bit)                (R)  ;/bit than hw2 each)
```

ATH:028080h/028054h/028080h - MAC_PCU_LAST_BEACON_TSF (R) ;hw2/hw4/hw6
ATH:028664h - MAC_PCU_LAST_BEACON2_TSF ;extra beacon (R) ;hw6 only

```
  0-31   VALUE (hw2: unspecified/LAST_TSTP)          (R)
```

ATH:028084h/028058h/028084h - MAC_PCU_NAV ;hw2/hw4/hw6

```
  0-25   VALUE (hw2: unspecified/NAV)
```

ATH:028088h/02805Ch/028088h - MAC_PCU_RTS_SUCCESS_CNT (R) ;hw2/hw4/hw6
ATH:02808Ch/028060h/02808Ch - MAC_PCU_RTS_FAIL_CNT (R) ;hw2/hw4/hw6
ATH:028090h/028064h/028090h - MAC_PCU_ACK_FAIL_CNT (R) ;hw2/hw4/hw6
ATH:028094h/028068h/028094h - MAC_PCU_FCS_FAIL_CNT (R) ;hw2/hw4/hw6
ATH:028098h/02806Ch/028098h - MAC_PCU_BEACON_CNT (R) ;hw2/hw4/hw6
ATH:02809Ch - MAC_PCU_BEACON2_CNT (R) ;hw6 only

```
  0-15   VALUE (COUNT or so?) (hw2: unspecified)     (R)
```

ATH:0280C0h/028070h/0280C0h - MAC_PCU_XRMODE ;hw2/hw4/hw6

```
  0-5    POLL_TYPE             (hw2: ini:1Ah)
  7      WAIT_FOR_POLL         (hw2: ini:0)
  20-31  FRAME_HOLD            (hw2: ini:680 decimal)
```

ATH:0280C4h/028074h/0280C4h - MAC_PCU_XRDEL ;hw2/hw4/hw6

```
  0-15   SLOT_DELAY            (hw2: ini:360 (decimal)
  16-31  CHIRP_DATA_DELAY      (hw2: ini:1680 decimal)
```

ATH:0280C8h/028078h/0280C8h - MAC_PCU_XRTO ;hw2/hw4/hw6

```
  0-15   CHIRP_TIMEOUT         (hw2: ini:7200 decimal)
  16-31  POLL_TIMEOUT          (hw2: ini:5000 decimal)
```

ATH:0280CCh/02807Ch/0280CCh - MAC_PCU_XRCRP ;hw2/hw4/hw6

```
  0      SEND_CHIRP            (hw2: ini:0)
  16-31  CHIRP_GAP             (hw2: ini:500 decimal)
```

ATH:0280D0h/028080h/0280D0h - MAC_PCU_XRSTMP ;hw2/hw4/hw6

```
  0      RX_ABORT_RSSI         (hw2: ini:0)
  1      RX_ABORT_BSSID        (hw2: ini:0)
  2      TX_STOMP_RSSI         (hw2: ini:0)
  3      TX_STOMP_BSSID        (hw2: ini:0)
  4      TX_STOMP_DATA         (hw2: ini:0)
```

```
  5      RX_ABORT_DATA         (hw2: ini:0)
  8-15   TX_STOMP_RSSI_THRESH  (hw2: ini:25h)
  16-23  RX_ABORT_RSSI_THRESH  (hw2: ini:25h)
```

ATH:0280E0h/028084h/0280E0h - MAC_PCU_ADDR1_MASK_L32 ;hw2/hw4/hw6
ATH:0280E4h/028088h/0280E4h - MAC_PCU_ADDR1_MASK_U16 ;hw2/hw4/hw6

```
  0-47   VALUE   (aka hw2:BSSID_MASK, ini:FFFFFFFFFFFFh)
```

This should be 48bit (as so in hw2/hw6, though hw4 claims 64bit, bit0-63).

ATH:0280E8h/02808Ch/0280E8h - MAC_PCU_TPC ;hw2/hw4/hw6

```
  0-5    ACK_PWR   (hw2: ini:3Fh)
  8-13   CTS_PWR   (hw2: ini:3Fh)
  16-21  CHIRP_PWR (hw2: ini:3Fh)
  24-29  hw6: RPT_PWR                                  ;-hw6 only
```

ATH:0280ECh/028090h/0280ECh - MAC_PCU_TX_FRAME_CNT ;hw2/hw4/hw6
ATH:0280F0h/028094h/0280F0h - MAC_PCU_RX_FRAME_CNT ;hw2/hw4/hw6
ATH:0280F4h/028098h/0280F4h - MAC_PCU_RX_CLEAR_CNT ;hw2/hw4/hw6
ATH:0280F8h/02809Ch/0280F8h - MAC_PCU_CYCLE_CNT ;hw2/hw4/hw6

```
  0-31   VALUE (aka COUNT or so?) (aka hw2: CNT, ini:0)
```

ATH:0280FCh/0280A0h/0280FCh - MAC_PCU_QUIET_TIME_1 ;hw2/hw4/hw6

```
  0-15   hw2: NEXT_QUIET   (hw2: ini:0)               ;\hw2 only (not hw4/hw6)
  16     hw2: QUIET_ENABLE (hw2: ini:0)               ;/
  17     ACK_CTS_ENABLE    (hw2: ini:1)               ;-hw2/hw4/hw6
```

ATH:028100h/0280A4h/028100h - MAC_PCU_QUIET_TIME_2 ;hw2/hw4/hw6

```
  0-15   hw2: QUIET_PERIOD           (hw2: ini:0002h) ;\differs in
  0-15   hw4: -                                       ; hw2, hw4, hw6
  0-15   hw6: DURATION2         ;-hw6 only            ;/
  16-31  DURATION (aka hw2: QUIET_DURATION, ini:0001h) ;-hw2/hw/hw6
```

ATH:028108h/0280A8h/028108h - MAC_PCU_QOS_NO_ACK ;hw2/hw4/hw6

```
  0-3    TWO_BIT_VALUES (hw2: NOACK_2_BIT_VALUES, ini:2)
  4-6    BIT_OFFSET     (hw2: NOACK_BIT_OFFSET, ini:5)
  7-8    BYTE_OFFSET    (hw2: NOACK_BYTE_OFFSET, ini:0)
```

ATH:02810Ch/0280ACh/02810Ch - MAC_PCU_PHY_ERROR_MASK ;hw2/hw4/hw6

```
  0-31   VALUE          (hw2: PHYERR_MASK, ini:0)
```

ATH:028110h/0280B0h/028110h - MAC_PCU_XRLAT ;hw2/hw4/hw6

```
  0-11   VALUE          (hw2: XR_TX_DELAY, ini:168h)
```

ATH:0280B4h/028114h - MAC_PCU_RXBUF ;hw4/hw6 (not hw2)

```
  0-10   HIGH_PRIORITY_THRSHD                         ;\hw4/hw6 only (not hw2)
  11     REG_RD_ENABLE                                ;/
```

Note: hw2 does have a "MAC_PCU_REG_TSF" register in this place, which seems to be something different than the hw4/hw6 stuff (unless the hw4/hw6 "RXBUF" is related to the hw2 "ACKSIFSMEM" array?).

ATH:028118h/0280B8h/028118h - MAC_PCU_MIC_QOS_CONTROL ;hw2/hw4/hw6

```
  0-15   VALUE_0..7 (2bit each)  (aka hw2: MICQOSCTL, ini:00AAh)
  16     ENABLE                  (aka hw2: MICQOSCTL_ENABLE, ini:1)
```

ATH:02811Ch/0280BCh/02811Ch - MAC_PCU_MIC_QOS_SELECT ;hw2/hw4/hw6

```
  0-31   VALUE_0..7 (4bit each)  (aka hw2: MICQOSSEL, ini:00003210h)
```

ATH:028124h/0280C4h/028124h - MAC_PCU_FILTER_OFDM_CNT ;hw2/hw4/hw6
ATH:028128h/0280C8h/028128h - MAC_PCU_FILTER_CCK_CNT ;hw2/hw4/hw6

```
  0-23   VALUE (count or so?)    (hw2: CNT, ini:0)
```

ATH:02812Ch/0280CCh/02812Ch - MAC_PCU_PHY_ERR_CNT_1 ;hw2/hw4/hw6
ATH:028134h/0280D4h/028134h - MAC_PCU_PHY_ERR_CNT_2 ;hw2/hw4/hw6
ATH:028168h/0280E4h/028168h - MAC_PCU_PHY_ERR_CNT_3 ;hw2/hw4/hw6
```
  0-23   VALUE (count or so?)    (hw2: PHYCNT, ini:0)
```

ATH:028130h/0280D0h/028130h - MAC_PCU_PHY_ERR_CNT_1_MASK ;hw2/hw4/hw6
ATH:028138h/0280D8h/028138h - MAC_PCU_PHY_ERR_CNT_2_MASK ;hw2/hw4/hw6
ATH:02816Ch/0280E8h/02816Ch - MAC_PCU_PHY_ERR_CNT_3_MASK ;hw2/hw4/hw6
```
  0-31   VALUE (mask or so?)     (hw2: PHYCNTMASK, ini:0)
```

ATH:02813Ch/0280DCh/005144h - MAC_PCU_TSF_THRESHOLD ;hw2/hw4/hw6
```
  0-15   VALUE                   (hw2: TSFTHRESH, ini:FFFFh)
```
Observe that hw2/hw6 port addresses are different here (unlike as usually).

ATH:028144h/0280E0h/028144h - MAC_PCU_PHY_ERROR_EIFS_MASK ;hw2/hw4/hw6
```
  0-31   VALUE                   (hw2: MASK, ini:0)
```

Misc Mode

ATH:028120h/0280C0h/028120h - MAC_PCU_MISC_MODE ;hw2/hw4/hw6
```
  0      BSSID_MATCH_FORCE              (hw2: ini:0)
  1      hw2: ACKSIFS_MEMORY_RESERVED   (hw2: ini:0)
  1      hw4/hw6: DEBUG_MODE_AD
  2      MIC_NEW_LOCATION(_ENABLE)      (hw2: ini:0)
  3      TX_ADD_TSF                     (hw2: ini:0)
  4      CCK_SIFS_MODE                  (hw2: ini:0)
  5      hw2: BFCOEF_MODE_RESERVED      (hw2: ini:0)
  6      hw2: BFCOEF_ENABLE             (hw2: ini:0)
  7      hw2: BFCOEF_UPDATE_SELF_GEN    (hw2: ini:1)
  8      hw2: BFCOEF_MCAST              (hw2: ini:1)
  9      hw2: DUAL_CHAIN_ANT_MODE       (hw2: ini:0)
  10     hw2: FALCON_DESC_MODE          (hw2: ini:0)
  5-8    hw4: -
  5      hw6: RXSM2SVD_PRE_RST
  6      hw6: RCV_DELAY_SOUNDING_IM_TXBF
  7-8    hw6: -
  9      hw4/hw6: DEBUG_MODE_BA_BITMAP
  10     hw4/hw6: DEBUG_MODE_SIFS
  11     KC_RX_ANT_UPDATE               (hw2: ini:1)
  12     TXOP_TBTT_LIMIT(_ENABLE)       (hw2: ini:0)
  13     hw2: FALCON_BB_INTERFACE       (hw2: ini:0)
  14     MISS_BEACON_IN_SLEEP           (hw2: ini:1)
  15-16  -
  17     hw2: BUG_12306_FIX_ENABLE      (hw2: ini:1)
  18     FORCE_QUIET_COLLISION          (hw2: ini:0)
  19     hw2: BUG_12549_FORCE_TXBF      (hw2: ini:0)
  20     BT_ANT_PREVENTS_RX             (hw2: ini:1)
  21     TBTT_PROTECT                   (hw2: ini:1)
  22     HCF_POLL_CANCELS_NAV           (hw2: ini:1)
  23     RX_HCF_POLL_ENABLE             (hw2: ini:1)
  24     CLEAR_VMF                      (hw2: ini:0)
  25     CLEAR_FIRST_HCF                (hw2: ini:0)
  26     hw2: ADHOC_MCAST_KEYID_ENABLE  (hw2: ini:0)
  27     hw2: ALLOW_RAC                 (hw2: ini:0)
  28-31  hw2: -
  26     hw4/hw6: CLEAR_BA_VALID
  27     hw4/hw6: SEL_EVM
  28     hw4/hw6: ALWAYS_PERFORM_KEY_SEARCH
  29     hw4/hw6: USE_EOP_PTR_FOR_DMA_WR
  30-31  hw4/hw6: DEBUG_MODE
```

ATH:02825Ch/028144h/028344h - MAC_PCU_MISC_MODE2 ;hw2/hw4/hw6

```
0      hw2: MGMT_CRYPTO_ENABLE        (ini:0) ;moved to bit1 in hw4 ;\
1      hw2: NO_CRYPTO_FOR_NON_DATA_PKT(ini:0) ;moved to bit2 in hw4 ; hw2
2-7    hw2: RESERVED                                                ;/
0      hw4/hw6: BUG_21532_FIX_ENABLE                                ;\
1      hw4/hw6: MGMT_CRYPTO_ENABLE                                  ; hw4/hw6
2      hw4/hw6: NO_CRYPTO_FOR_NON_DATA_PKT                          ;/
3      hw4: RESERVED
3      hw6: BUG_58603_FIX_ENABLE                                    ;-hw6
4      hw6 and hw4.2: BUG_58057_FIX_ENABLE ;-hw4.2 and up (not hw2 and hw4.0)
5      hw4/hw6: RESERVED                                            ;\
6      hw4/hw6: ADHOC_MCAST_KEYID_ENABLE                            ; hw4/hw6
7      hw4/hw6: CFP_IGNORE                                          ;/
8-15   MGMT_QOS                                                     ;-all hw
16     hw2: BC_MC_WAPI_MODE          (ini:0) ;moved to bit18 in hw4 ;\
17     hw2: IGNORE_TXOP_FOR_1ST_PKT  (ini:0) ;moved to bit22 in hw4 ; hw2
18     hw2: IGNORE_TXOP_IF_ZERO      (ini:0) ;moved to bit23 in hw4 ;
19-31  hw2_ RESERVED                                               ;/
16     hw4/hw6: ENABLE_LOAD_NAV_BEACON_DURATION                     ;\
17     hw4/hw6: AGG_WEP                                             ;
18     hw4/hw6: BC_MC_WAPI_MODE                                     ;
19     hw4/hw6: DUR_ACCOUNT_BY_BA                                   ; hw4/hw6
20     hw4/hw6: BUG_28676                                           ;
21     hw4/hw6: CLEAR_MORE_FRAG                                     ;
22     hw4/hw6: IGNORE_TXOP_1ST_PKT                                 ;/
23     hw4: IGNORE_TXOP_IF_ZERO  ;moved to MISC_MODE3.bit22 in hw6  ;\
24     hw4: PM_FIELD_FOR_DAT     ;moved to MISC_MODE3.bit24 in hw6  ;
25     hw4: PM_FIELD_FOR_MGMT    ;moved to MISC_MODE3.bit25 in hw6  ; hw4
26     hw4: BEACON_FROM_TO_DS    ;moved to MISC_MODE3.bit23 in hw6? ;
27     hw4: RCV_TIMESTAMP_FIX    ;moved to bit25 in hw6             ;
28-31  hw4: RESERVED                                               ;/
23     hw6: MPDU_DENSITY_STS_FIX                                    ;\
24     hw6: MPDU_DENSITY_WAIT_WEP                                   ;
25     hw6: RCV_TIMESTAMP_FIX    ;moved from bit27 in hw4           ;
27     hw6: DECOUPLE_DECRYPTION                                     ; hw6
28     hw6: H_TO_SW_DEBUG_MODE                                      ;
29     hw6: TXBF_ACT_RPT_DONE_PASS                                  ;
30     hw6: PCU_LOOP_TXBF                                           ;
31     hw6: CLEAR_WEP_TXBUSY_ON_TXURN                               ;/
```

Observe that hw2/hw6 port addresses are different here (unlike as usually).

ATH:0283D0h - MAC_PCU_MISC_MODE3 ;hw6

```
0      BUG_55702_FIX_ENABLE
1      AES_3STREAM
2      REGULAR_SOUNDING
3      BUG_58011_FIX_ENABLE
4      BUG_56991_FIX_ENABLE
5      WOW_ADDR1_MASK_ENABLE
6      BUG_61936_FIX_ENABLE
7      CHECK_LENGTH_FOR_BA
8-15   BA_FRAME_LENGTH
16     MATCH_TID_FOR_BA
17     WAPI_ORDER_MASK
18     BB_LDPC_EN
19     SELF_GEN_SMOOTHING
20     SMOOTHING_FORCE
21     ALLOW_RAC
22     IGNORE_TXOP_IF_ZER0  ;uh, ZerNull or Zero?   ;moved from MODE2.bit23
23     BEACON_FROM_TO_DS_CHECK                      ;moved from MODE2.bit26?
24     PM_FIELD_FOR_DAT                             ;moved from MODE2.bit24
25     PM_FIELD_FOR_MGMT                            ;moved from MODE2.bit25
26     PM_FIELD2_FOR_CTL
27     PM_FIELD2_FOR_DAT
```

```
  28     PM_FIELD2_FOR_MGT
  29     KEY_MISS_FIX
  30     PER_STA_WEP_ENTRY_ENABLE
  31     TIME_BASED_DISCARD_EN
```

ATH:0283D4h - MAC_PCU_MISC_MODE4 ;hw6

```
  0      BC_MC_WAPI_MODE2_EN
  1      BC_MC_WAPI_MODE2
  2      SYNC_TSF_ON_BEACON
  3      SYNC_TSF_ON_BCAST_PROBE_RESP
  4      SYNC_TSF_ON_MCAST_PROBE_RESP
  5      SYNC_TSF_ON_UCAST_MOON_PROBE_RESP
  6      SYNC_TSF_ON_UCAST_PROBE_RESP
```

_____ Basic Rate Set _____

ATH:0282A4h - MAC_PCU_BASIC_RATE_SET0 ;hw2 (other as in hw4/hw6)
ATH:0282A8h - MAC_PCU_BASIC_RATE_SET1 ;hw2 (other as in hw4/hw6)
ATH:0282ACh - MAC_PCU_BASIC_RATE_SET2 ;hw2 (other as in hw4/hw6)

```
 Bitfields for hw2 RATE_SET0 register:
  0-4    BRATE_1MB_L             (hw2: ini:#CCK_RATE_1Mb_L)
  5-9    BRATE_2MB_L             (hw2: ini:#CCK_RATE_2Mb_L)
  10-14  BRATE_2MB_S             (hw2: ini:#CCK_RATE_2Mb_S)
  15-19  BRATE_5_5MB_L           (hw2: ini:#CCK_RATE_5_5Mb_L)
  20-24  BRATE_5_5MB_S           (hw2: ini:#CCK_RATE_5_5Mb_S)
  25-29  BRATE_11MB_L            (hw2: ini:#CCK_RATE_11Mb_L)
 Bitfields for hw2 RATE_SET1 register:
  0-4    BRATE_11MB_S            (hw2: ini:#CCK_RATE_11Mb_S)
  5-9    BRATE_6MB               (hw2: ini:#OFDM_RATE_6Mb)
  10-14  BRATE_9MB               (hw2: ini:#OFDM_RATE_6Mb, too?)
  15-19  BRATE_12MB              (hw2: ini:#OFDM_RATE_12Mb)
  20-24  BRATE_18MB              (hw2: ini:#OFDM_RATE_12Mb, too?)
  25-29  BRATE_24MB              (hw2: ini:#OFDM_RATE_24Mb)
 Bitfields for hw2 RATE_SET2 register:
  0-4    BRATE_36MB              (hw2: ini:#OFDM_RATE_24Mb, too?)
  5-9    BRATE_48MB              (hw2: ini:#OFDM_RATE_24Mb, too?)
  10-14  BRATE_54MB              (hw2: ini:#OFDM_RATE_24Mb, too?)
```

Alongsides, hw2 source code defines following "Rate" values:

```
  OFDM_RATE_6Mb    = 0Bh     CCK_RATE_1Mb_L   = 1Bh      XR_RATE_0_25Mb   = 03h
  OFDM_RATE_9Mb    = 0Fh     CCK_RATE_2Mb_L   = 1Ah      XR_RATE_0_5Mb    = 07h
  OFDM_RATE_12Mb   = 0Ah     CCK_RATE_2Mb_S   = 1Eh      XR_RATE_1Mb      = 02h
  OFDM_RATE_18Mb   = 0Eh     CCK_RATE_5_5Mb_L = 19h      XR_RATE_2Mb      = 06h
  OFDM_RATE_24Mb   = 09h     CCK_RATE_5_5Mb_S = 1Dh      XR_RATE_3Mb      = 01h
  OFDM_RATE_36Mb   = 0Dh     CCK_RATE_11Mb_L  = 18h      (the XR_stuff might be
  OFDM_RATE_48Mb   = 08h     CCK_RATE_11Mb_S  = 1Ch      unrelated to RATE_SET)
  OFDM_RATE_54Mb   = 0Ch
```

Note: The hw2 RATE_SET registers contain 75bit (30+30+15bit), hw4/hw6 has similar RATE_SET registers with 100bit (4x25bit), that registers may have similar functions - but their content is obviously differently arranged.

ATH:028328h/0283E0h - MAC_PCU_BASIC_RATE_SET0 ;hw4/hw6 (other as in hw2)
ATH:02832Ch/0283E4h - MAC_PCU_BASIC_RATE_SET1 ;hw4/hw6 (other as in hw2)
ATH:028330h/0283E8h - MAC_PCU_BASIC_RATE_SET2 ;hw4/hw6 (other as in hw2)
ATH:028334h/0283ECh - MAC_PCU_BASIC_RATE_SET3 ;hw4/hw6 (new in hw4/hw6)

```
  0-24   VALUE (maybe this 25bit value is meant to contain 5 rates of 5bit ?)
```

Note: The hw4/hw6 RATE_SET registers contain 100bit (4x25bit), hw2 has similar RATE_SET registers with 75bit (30+30+15bit), that registers may have similar functions - but their content is obviously differently arranged.

_____ Bluetooth Mode _____

ATH:028170h/0280ECh/028170h - MAC_PCU_BLUETOOTH_MODE ;hw2/hw4/hw6

```
  0-7    TIME_EXTEND           (hw2: ini:20h)
  8      TX_STATE_EXTEND       (hw2: ini:1)
  9      TX_FRAME_EXTEND       (hw2: ini:1)
  10-11  MODE                  (hw2: ini:3)
  12     QUIET                 (hw2: ini:1)
  13-16  QCU_THRESH            (hw2: ini:1)
  17     RX_CLEAR_POLARITY     (hw2: ini:0)
  18-23  PRIORITY_TIME         (hw2: ini:05h)
  24-31  FIRST_SLOT_TIME       (hw2: ini:9Bh)
```

ATH:02817Ch/0280F4h/02817Ch - MAC_PCU_BLUETOOTH_MODE2 ;hw2/hw4/hw6

```
  0-7    BCN_MISS_THRESH (hw2: ini:0)
  8-15   BCN_MISS_CNT                          (R)
  16     HOLD_RX_CLEAR (hw2: ini:0)
  17     SLEEP_ALLOW_BT_ACCESS (hw2: WL_CONTROL_ANT, ini:0)
  18     hw2: RESPOND_TO_BT_ACTIVE (hw2: ini:0)                ;-hw2 only
  19     PROTECT_BT_AFTER_WAKEUP (hw2: ini:0)
  20     DISABLE_BT_ANT (hw2: ini:0)
  21     hw4/hw6: QUIET_2_WIRE                                 ;\
  22-23  hw4/hw6: WL_ACTIVE_MODE                               ;
  24     hw4/hw6: WL_TXRX_SEPARATE                             ;
  25     hw4/hw6: RS_DISCARD_EXTEND                            ; hw4/hw6 only
  26-27  hw4/hw6: TSF_BT_ACTIVE_CTRL                           ;
  28-29  hw4/hw6: TSF_BT_PRIORITY_CTRL                         ;
  30     hw4/hw6: INTERRUPT_ENABLE                             ;
  31     hw4/hw6: PHY_ERR_BT_COLL_ENABLE                       ;/
```

ATH:028164h/0281D4h - MAC_PCU_BLUETOOTH_MODE3 ;hw4/hw6 (not hw2)

```
  0-7    WL_ACTIVE_TIME                                ;\
  8-15   WL_QC_TIME                                    ;
  16-19  ALLOW_CONCURRENT_ACCESS                       ;
  20     hw4: SHARED_RX                    ;<-- hw4    ;
  20     hw6: AGC_SATURATION_CNT_ENABLE    ;<-- hw6    ; hw4/hw6 only (not hw2)
  21     WL_PRIORITY_OFFSET_EN                         ;
  22     RFGAIN_LOCK_SRC                               ;
  23     DYNAMIC_PRI_EN                                ;
  24     DYNAMIC_TOGGLE_WLA_EN                         ;
  25-26  SLOT_SLOP                                     ;
  27     BT_TX_ON_EN                                   ;
  28-31  BT_PRIORITY_EXTEND_THRES                      ;/
```

ATH:028168h/0281D8h - MAC_PCU_BLUETOOTH_MODE4 ;hw4/hw6 (not hw2)

```
  0-15   BT_ACTIVE_EXTEND                              ;\hw4/hw6 only (not hw2)
  16-31  BT_PRIORITY_EXTEND                            ;/
```

ATH:0281DCh - MAC_PCU_BLUETOOTH_MODE5 ;hw6 (not hw2/hw4)

```
  0-2    MCI_WL_LEVEL_MULT                             ;\
  3      TX_ON_SRC                                     ;
  4-19   TIMER_TARGET                                  ; hw6 only (not hw2/hw4)
  20     SHARED_RX                                     ;
  21     USE_BTP_EXT                                   ;/
```

ATH:028174h/0280F0h/0281E0h - MAC_PCU_BLUETOOTH_WEIGHTS ;hw2/hw4/hw6

```
  0-15   BT_WEIGHT             (hw2: ini:FA50h)
  16-31  WL_WEIGHT             (hw2: ini:FAA4h)
```

Observe that hw2/hw6 port addresses are different here (unlike as usually).
For hw4 only: There's also a "WL_WEIGHT_CONTD" in a "WEIGHTS" register.

ATH:028158h - MAC_PCU_BLUETOOTH_WEIGHTS2 ;hw4 only (not hw2/hw6)

```
  16-31  WL_WEIGHT_CONTD (extends "WL_WEIGHT" or so)   ;-hw4 only (not hw2/hw6)
```

```
_____________________________ hw2/hw6 only _____________________________
```

ATH:028178h/028154h/028178h - MAC_PCU_HCF_TIMEOUT ;hw2/hw6 (not hw4)
```
  0-15   VALUE      (hw2: TIMEOUT, ini:100h)          ;-hw2/hw6 only (not hw4)
```

ATH:0281D0h/0280F8h/0281D0h - MAC_PCU_TXSIFS ;hw2/hw6 (not hw4)
```
  0-7    SIFS_TIME              (hw2: ini: 16 decimal)
  8-11   TX_LATENCY             (hw2: ini:2)
  12-14  ACK_SHIFT              (hw2: ini:3)
```

ATH:0281ECh/0280FCh/0281ECh - MAC_PCU_TXOP_X ;hw2/hw6 (not hw4)
```
  0-7    VALUE                  (hw2: TXOP_X, ini:0)
```

ATH:0281F0h/028100h/0281F0h - MAC_PCU_TXOP_0_3 ;hw2/hw6 (not hw4)
ATH:0281F4h/028104h/0281F4h - MAC_PCU_TXOP_4_7 ;hw2/hw6 (not hw4)
ATH:0281F8h/028108h/0281F8h - MAC_PCU_TXOP_8_11 ;hw2/hw6 (not hw4)
ATH:0281FCh/02810Ch/0281FCh - MAC_PCU_TXOP_12_15 ;hw2/hw6 (not hw4)
```
  0-7    TXOP_0 / TXOP_4 / TXOP_8  / TXOP_12   (hw2: ini:0)
  8-15   TXOP_1 / TXOP_5 / TXOP_9  / TXOP_13   (hw2: ini:0)
  16-23  TXOP_2 / TXOP_6 / TXOP_10 / TXOP_14   (hw2: ini:0)
  24-31  TXOP_3 / TXOP_7 / TXOP_11 / TXOP_15   (hw2: ini:0)
```

```
_____________________________ hw4/hw6 only _____________________________
```

ATH:028304h/028024h - MAC_PCU_BT_WL_1 ;hw4/hw6
ATH:028308h/028028h - MAC_PCU_BT_WL_2 ;hw4/hw6
ATH:02830Ch/02802Ch - MAC_PCU_BT_WL_3 ;hw4/hw6
ATH:028310h/028030h - MAC_PCU_BT_WL_4 ;hw4/hw6
```
  0-31   WEIGHT
```

ATH:028314h/028034h - MAC_PCU_COEX_EPTA ;hw4/hw6 only
```
  0-5    LINKID
  6-12   WT_IDX
```

ATH:028300h/0281E4h - MAC_PCU_BT_BT_ASYNC ;hw4/hw6 (not hw2)
```
  0-3    TXHP_WEIGHT                          ;\
  4-7    TXLP_WEIGHT                          ; hw4/hw6 only (not hw2)
  8-11   RXHP_WEIGHT                          ;
  12-15  RXLP_WEIGHT                          ;/
```

ATH:028110h/028264h - MAC_PCU_LOGIC_ANALYZER ;hw4/hw6 (not hw2)
```
  0      HOLD                                 ;\
  1      CLEAR                                ;
  2      STATE                          (R)   ; hw4/hw6 only (not hw2)
  3      ENABLE                               ;
  4-7    QCU_SEL                              ;
  8-17   INT_ADDR                       (R)   ;
  18-31  DIAG_MODE                            ;/
```

ATH:028114h/028268h - MAC_PCU_LOGIC_ANALYZER_32L ;hw4/hw6 (not hw2)
ATH:028118h/02826Ch - MAC_PCU_LOGIC_ANALYZER_16U ;hw4/hw6 (not hw2)
```
  0-47   MASK                                 ;\hw4/hw6 only (not hw2)
  48-31  -                                    ;/
```

ATH:02815Ch/0281C8h - MAC_PCU_BLUETOOTH_TSF_BT_ACTIVE ;hw4/hw6 (not hw2)
```
  0-31   VALUE                          (R)   ;-hw4/hw6 only (not hw2)
```

ATH:028160h/0281CCh - MAC_PCU_BLUETOOTH_TSF_BT_PRIORITY ;hw4/hw6 (not hw2)

```
  0-31   VALUE                          (R)     ;-hw4/hw6 only (not hw2)
```

ATH:028318h/028150h - MAC_PCU_COEX_LNAMAXGAIN1 ;hw4/hw6 (not hw2)
ATH:02831Ch/028154h - MAC_PCU_COEX_LNAMAXGAIN2 ;hw4/hw6 (not hw2)
ATH:028320h/028158h - MAC_PCU_COEX_LNAMAXGAIN3 ;hw4/hw6 (not hw2)
ATH:028324h/02815Ch - MAC_PCU_COEX_LNAMAXGAIN4 ;hw4/hw6 (not hw2)

```
  0-7    MAXGAIN1     ;\that are 4 gain value          ;\
  8-15   MAXGAIN2     ; for each of the 4 registers    ; hw4/hw6 only (not hw2)
  16-23  MAXGAIN3     ; (ie. 16 values in total)       ;
  24-31  MAXGAIN4     ;/                               ;/
```

ATH:028050h/028070h - MAC_PCU_DYM_MIMO_PWR_SAVE ;hw4/hw6 only

```
  0      hw4/hw6: USE_MAC_CTRL                         ;\
  1      hw4/hw6: HW_CTRL_EN                           ;
  2      hw4/hw6: SW_CHAIN_MASK_SEL                    ; hw4/hw6 only
  4-6    hw4/hw6: LOW_PWR_CHAIN_MASK                   ;
  8-10   hw4/hw6: HI_PWR_CHAIN_MASK                    ;/
```

ATH:02811Ch/02829Ch - MAC_PCU_PHY_ERR_CNT_MASK_CONT ;hw4/hw6 (not hw2)

```
  0-7    MASK1
  8-15   MASK2
  16-23  MASK3
```

ATH:028120h/028300h - MAC_PCU_AZIMUTH_MODE ;hw4/hw6 (not hw2)

```
  0      DISABLE_TSF_UPDATE
  1      KEY_SEARCH_AD1
  2      TX_TSF_STATUS_SEL
  3      RX_TSF_STATUS_SEL
  4      CLK_EN
  5      TX_DESC_EN
  6      ACK_CTS_MATCH_TX_AD2
  7      BA_USES_AD1
  8      hw6: WMAC_CLK_SEL
  9      hw6: FILTER_PASS_HOLD
```

ATH:02814Ch/028314h - MAC_PCU_AZIMUTH_TIME_STAMP ;hw4/hw6 (not hw2)

```
  0-31   VALUE
```

ATH:028124h/028318h - MAC_PCU_20_40_MODE ;hw4/hw6 (not hw2)

```
  0      JOINED_RX_CLEAR
  1      EXT_PIFS_ENABLE
  2      TX_HT20_ON_EXT_BUSY
  3      SWAMPED_FORCES_RX_CLEAR_CTL_IDLE
  4-15   PIFS_CYCLES
```

ATH:028128h/028328h - MAC_PCU_RX_CLEAR_DIFF_CNT ;hw4/hw6 (not hw2)

```
  0-31   VALUE
```

ATH:02812Ch/02832Ch - MAC_PCU_SELF_GEN_ANTENNA_MASK ;hw4/hw6 (not hw2)

```
  0-2    VALUE
  3      hw6: ONE_RESP_EN
  4      hw6: FORCE_CHAIN_0
```

ATH:028130h/028330h - MAC_PCU_BA_BAR_CONTROL ;hw4/hw6 (not hw2)

```
  0-3    COMPRESSED_OFFSET
  4-7    ACK_POLICY_OFFSET
  8      COMPRESSED_VALUE
  9      ACK_POLICY_VALUE
  10     FORCE_NO_MATCH
  11     TX_BA_CLEAR_BA_VALID
  12     UPDATE_BA_BITMAP_QOS_NULL
```

ATH:028134h/028334h - MAC_PCU_LEGACY_PLCP_SPOOF ;hw4/hw6 (not hw2)

```
0-7    EIFS_MINUS_DIFS
8-12   MIN_LENGTH
```

ATH:028138h/028338h - MAC_PCU_PHY_ERROR_MASK_CONT ;hw4/hw6 (not hw2)

```
0-7    MASK_VALUE
16-23  EIFS_VALUE
24-31  hw6: AIFS_VALUE
```

ATH:02813Ch/02833Ch - MAC_PCU_TX_TIMER ;hw4/hw6 (not hw2)

```
0-14   TX_TIMER
15     TX_TIMER_ENABLE
16-19  RIFS_TIMER
20-24  QUIET_TIMER
25     QUIET_TIMER_ENABLE
```

ATH:028140h/028340h - MAC_PCU_TXBUF_CTRL ;hw4/hw6 (not hw2)

```
0-11   USABLE_ENTRIES
16     TX_FIFO_WRAP_ENABLE
```

ATH:028148h/028348h - MAC_PCU_ALT_AES_MUTE_MASK ;hw4/hw6 (not hw2)

```
16-31  QOS
```

ATH:028338h/028600h - MAC_PCU_RX_INT_STATUS0 ;hw4/hw6

```
0-7    FRAME_CONTROL_L          (R)
8-15   FRAME_CONTROL_H          (R)
16-23  DURATION_L               (R)
24-31  DURATION_H               (R)
```

ATH:02833Ch/028604h - MAC_PCU_RX_INT_STATUS1 ;hw4/hw6

```
0-17   VALUE                    (R)
```

ATH:028340h/028608h - MAC_PCU_RX_INT_STATUS2 ;hw4/hw6

```
0-26   VALUE                    (R)
```

ATH:028344h/02860Ch - MAC_PCU_RX_INT_STATUS3 ;hw4/hw6

```
0-23   VALUE                    (R)
```

ATH:028348h/028610h - HT_HALF_GI_RATE1 ;MCS0..3 ;hw4/hw6
ATH:02834Ch/028614h - HT_HALF_GI_RATE2 ;MCS4..7 ;hw4/hw6

```
0-7    MCS0 / MCS4
8-15   MCS1 / MCS5
16-23  MCS2 / MCS6
24-31  MCS3 / MCS7
```

ATH:028350h/028618h - HT_FULL_GI_RATE1 ;MCS0..3 ;hw4/hw6
ATH:028354h/02861Ch - HT_FULL_GI_RATE2 ;MCS4..7 ;hw4/hw6

```
0-7    MCS0 / MCS4
8-15   MCS1 / MCS5
16-23  MCS2 / MCS6
24-31  MCS3 / MCS7
```

ATH:028358h/028620h - LEGACY_RATE1 ;RATE 8..12 ;hw4/hw6
ATH:02835Ch/028624h - LEGACY_RATE2 ;RATE 13..15 and RATE 24..25 ;hw4/hw6
ATH:028360h/028628h - LEGACY_RATE3 ;RATE 26..30 ;hw4/hw6

```
0-5    RATE8  / RATE13 / RATE26
6-11   RATE9  / RATE14 / RATE27
12-17  RATE10 / RATE15 / RATE28
```

```
  18-23  RATE11 / RATE24 / RATE29
  24-29  RATE12 / RATE25 / RATE30
```

ATH:028364h/02862Ch - RX_INT_FILTER ;hw4/hw6

```
  0      ENABLE
  1      DIRECTED
  2      BCAST
  3      MCAST
  4      RTS
  5      ACK
  6      CTS
  7      RETRY
  8      MORE_DATA
  9      MORE_FRAG
  10     RATE_HIGH
  11     RATE_LOW
  12     RSSI
  13     LENGTH_HIGH
  14     LENGTH_LOW
  15     EOSP
  16     AMPDU
  17     hw4.2: BEACON                 ;-hw6 and newer "hw4.2" revision only
  18     hw6:   RSSI_HIGH              ;-hw6 only
```

ATH:028368h/028630h - RX_INT_OVERFLOW ;hw4/hw6

```
  0      STATUS
```

ATH:02836Ch/028634h - RX_FILTER_THRESH0 ;hw4/hw6

```
  0-7    RATE_HIGH
  8-15   RATE_LOW
  16-23  RSSI_LOW
  24-31  hw6: RSSI_HIGH
```

ATH:028370h/028638h - RX_FILTER_THRESH1 ;hw4/hw6

```
  0-11   LENGTH_HIGH
  12-23  LENGTH_LOW
```

ATH:028374h/02863Ch - RX_PRIORITY_THRESH0 ;hw4/hw6

```
  0-7    RATE_HIGH
  8-15   RATE_LOW
  16-23  RSSI_HIGH
  24-31  RSSI_LOW
```

ATH:028378h/028640h - RX_PRIORITY_THRESH1 ;hw4/hw6

```
  0-11   LENGTH_HIGH
  12-23  LENGTH_LOW
  24-31  XCAST_RSSI_HIGH
```

ATH:02837Ch/028644h - RX_PRIORITY_THRESH2 ;hw4/hw6

```
  0-7    PRESP_RSSI_HIGH
  8-15   MGMT_RSSI_HIGH
  16-23  BEACON_RSSI_HIGH
  24-31  NULL_RSSI_HIGH
```

ATH:028380h/028648h - RX_PRIORITY_THRESH3 ;hw4/hw6

```
  0-7    PREQ_RSSI_HIGH
  8-15   PS_POLL_RSSI_HIGH
```

ATH:028384h/02864Ch - RX_PRIORITY_OFFSET0 ;hw4/hw6

```
  0-5    PHY_RATE_HIGH
  6-11   PHY_RATE_LOW
```

```
  12-17  RSSI_HIGH
  18-23  RSSI_LOW
  24-29  XCAST_RSSI_HIGH
```

ATH:028388h/028650h - RX_PRIORITY_OFFSET1 ;hw4/hw6
```
  0-5    LENGTH_HIGH
  6-11   LENGTH_LOW
  12-17  PRESP_RSSI_HIGH
  18-23  RETX
  24-29  RTS
```

ATH:02838Ch/028654h - RX_PRIORITY_OFFSET2 ;hw4/hw6
```
  0-5    XCAST
  6-11   PRESP
  12-17  ATIM
  18-23  MGMT
  24-29  BEACON
```

ATH:028390h/028658h - RX_PRIORITY_OFFSET3 ;hw4/hw6
```
  0-5    MORE
  6-11   EOSP
  12-17  AMPDU
  18-23  AMSDU
  24-29  PS_POLL
```

ATH:028394h/02865Ch - RX_PRIORITY_OFFSET4 ;hw4/hw6
```
  0-5    PREQ
  6-11   NULL
  12-17  BEACON_SSID
  18-23  MGMT_RSSI_HIGH
  24-29  BEACON_RSSI_HIGH
```

ATH:028398h/028660h - RX_PRIORITY_OFFSET5 ;hw4/hw6
```
  0-5    NULL_RSSI_HIGH
  6-11   PREQ_RSSI_HIGH
  12-17  PS_POLL_RSSI_HIGH
```

ATH:028200h..0282FCh - MAC_PCU_BT_BT[0..63] ;hw4 (at 028200h)
ATH:028500h..0285FCh - MAC_PCU_BT_BT[0..63] ;hw6 (at 028500h)
```
  0-31   WEIGHT
```

ATH:028400h..0284FCh - MAC_PCU_TXBUF_BA[0..63] ;hw4/hw6
```
  0-31   DATA
```

ATH:028800h..028BFCh - MAC_PCU_KEY_CACHE_1[0..255] ;hw4 (256 words)
ATH:028800h..028FFCh - MAC_PCU_KEY_CACHE[0..511] ;hw6 (512 words)
```
  0-31   DATA
```

ATH:02E000h..02E7FCh - MAC_PCU_BUF[0..511] ;hw4 (512 words)
ATH:02E000h..02FFFCh - MAC_PCU_BUF[0..2047] ;hw6 (2048 words)
```
  0-31   DATA
```

```
 ____________________________________ hw6 only ____________________________________
```

ATH:0280A0h - MAC_PCU_BASIC_SET ;hw6 only
```
  0-31   MCS                                             ;-hw6 only
```

ATH:0280A4h - MAC_PCU_MGMT_SEQ ;hw6 only
```
  0-11   MIN                                             ;\hw6 only
  16-27  MAX                                             ;/
```

ATH:0280A8h - MAC_PCU_BF_RPT1 ;hw6 only

```
  0-7    V_ACTION_VALUE                          ;\
  8-15   CV_ACTION_VALUE                         ; hw6 only
  16-23  CATEGORY_VALUE                          ;
  24-27  FRAME_SUBTYPE_VALUE                     ;
  28-29  FRAME_TYPE_VALUE                        ;/
```

ATH:0280ACh - MAC_PCU_BF_RPT2 ;hw6 only

```
  0-3    FRAME_SUBTYPE_VALUE                     ;-hw6 only
```

ATH:0280B0h - MAC_PCU_TX_ANT_1 ;hw6 only
ATH:0280B4h - MAC_PCU_TX_ANT_2 ;hw6 only
ATH:0280B8h - MAC_PCU_TX_ANT_3 ;hw6 only
ATH:0280BCh - MAC_PCU_TX_ANT_4 ;hw6 only

```
  0-31   VALUE                                   ;-hw6 only
```

ATH:028038h - MAC_PCU_MAX_CFP_DUR ;hw6 only (merged two hw4 registers here)

```
  0-15   VALUE                  ;-formerly bit0-15 of "PCU_MAX_CFP_DUR"
  16-19  USEC_FRAC_NUMERATOR    ;-formerly bit0-3 of "MAC_PCU_MAX_CFP_DUR"
  24-27  USEC_FRAC_DENOMINATOR  ;-formerly bit4-7 of "MAC_PCU_MAX_CFP_DUR"
```

ATH:028020h - MAC_PCU_BCN_RSSI_CTL2 ;hw6 only

```
  0-7    RSSI2_LOW_THRESH
  16-23  RSSI2_HIGH_THRESH
  29     RESET2
```

ATH:0280DCh - MAC_PCU_SELF_GEN_DEFAULT ;hw6 only

```
  0-2    MMSS                                    ;\
  3-4    CEC                                     ; hw6 only
  5      STAGGER_SOUNDING                        ;/
```

ATH:02831Ch - MAC_PCU_H_XFER_TIMEOUT ;hw6 only

```
  0-4    VALUE                                   ;\
  5      DISABLE                                 ;
  6      EXTXBF_IMMEDIATE_RESP                   ; hw6 only
  7      DELAY_EXTXBF_ONLY_UPLOAD_H              ;
  8      EXTXBF_NOACK_NORPT                      ;/
```

ATH:028200h - MAC_PCU_TDMA_TXFRAME_START_TIME_TRIGGER_LSB ;hw6

```
  0-31   VALUE
```

ATH:028204h - MAC_PCU_TDMA_TXFRAME_START_TIME_TRIGGER_MSB ;hw6

```
  0-31   VALUE
```

ATH:028208h - MAC_PCU_TDMA_SLOT_ALERT_CNTL ;hw6

```
  0-15   VALUE
```

ATH:028284h - PCU_1US ;hw6

```
  0-6    SCALER
```

ATH:028288h - PCU_KA ;hw6

```
  0-11   DEL
```

ATH:028350h - ASYNC_FIFO_REG1 ;hw6

```
  0-29   DBG
```

ATH:028354h - ASYNC_FIFO_REG2 ;hw6

```
  0-27   DBG
```

ATH:028358h - ASYNC_FIFO_REG3 ;hw6

0-9	DBG
10	DATAPATH_SEL
31	SFT_RST_N

ATH:028388h - MAC_PCU_LOCATION_MODE_CONTROL ;hw6

0	ENABLE
1	UPLOAD_H_DISABLE

ATH:02838Ch - MAC_PCU_LOCATION_MODE_TIMER ;hw6

0-31	VALUE

ATH:0283A0h - MAC_PCU_DIRECT_CONNECT ;hw6

0	TSF2_ENABLE
1	TS_TSF_SEL
2	TSF1_UPDATE
3	TSF2_UPDATE
4	MY_BEACON_OVERRIDE
5	MY_BEACON2_OVERRIDE
6	BMISS_CNT_TSF_SEL
7	BMISS_CNT_OVERRIDE
8-31	RESERVED

ATH:0283A4h - MAC_PCU_TID_TO_AC ;hw6

0-31	DATA

ATH:0283A8h - MAC_PCU_HP_QUEUE ;hw6

0	ENABLE
1	AC_MASK_BE
2	AC_MASK_BK
3	AC_MASK_VI
4	AC_MASK_VO
5	HPQON_UAPSD
6	FRAME_FILTER_ENABLE0
7	FRAME_BSSID_MATCH0
8-9	FRAME_TYPE0
10-11	FRAME_TYPE_MASK0
12-15	FRAME_SUBTYPE0
16-19	FRAME_SUBTYPE_MASK0
20	UAPSD_EN
21	PM_CHANGE
22	NON_UAPSD_EN
23	UAPSD_AC_MUST_MATCH
24	UAPSD_ONLY_QOS

ATH:0283BCh - MAC_PCU_AGC_SATURATION_CNT0 ;hw6
ATH:0283C0h - MAC_PCU_AGC_SATURATION_CNT1 ;hw6
ATH:0283C4h - MAC_PCU_AGC_SATURATION_CNT2 ;hw6

0-31	VALUE

ATH:0283C8h - MAC_PCU_HW_BCN_PROC1 ;hw6

0	CRC_ENABLE
1	RESET_CRC
2	EXCLUDE_BCN_INTVL
3	EXCLUDE_CAP_INFO
4	EXCLUDE_TIM_ELM
5	EXCLUDE_ELM0
6	EXCLUDE_ELM1

```
  7       EXCLUDE_ELM2
  8-15    ELM0_ID
  16-23   ELM1_ID
  24-31   ELM2_ID
```

ATH:0283CCh - MAC_PCU_HW_BCN_PROC2 ;hw6

```
  0       FILTER_INTERVAL_ENABLE
  1       RESET_INTERVAL
  2       EXCLUDE_ELM3
  8-15    FILTER_INTERVAL
  16-23   ELM3_ID
```

ATH:0283DCh - MAC_PCU_PS_FILTER ;hw6

```
  0       ENABLE
  1       PS_SAVE_ENABLE
```

ATH:028668h - MAC_PCU_PHY_ERROR_AIFS ;hw6

```
  0       MASK_ENABLE
```

ATH:02866Ch - MAC_PCU_PHY_ERROR_AIFS_MASK ;hw6

```
  0-31    VALUE
```

ATH:028670h - MAC_PCU_FILTER_RSSI_AVE ;hw6

```
  0-7     AVE_VALUE
  8-10    NUM_FRAMES_EXPONENT
  11      ENABLE
  12      RESET
```

ATH:028674h - MAC_PCU_TBD_FILTER ;hw6

```
  0       USE_WBTIMER_TX_TS
  1       USE_WBTIMER_RX_TS
```

ATH:028678h - MAC_PCU_BT_ANT_SLEEP_EXTEND ;hw6

```
  0-15    CNT
```

_____ Wake on Wireless (WOW) hw6 only _____

ATH:02825Ch - MAC_PCU_WOW1 ;WOW Misc ;hw6

```
  0-7     PATTERN_ENABLE
  8-15    PATTERN_DETECT                (R)
  16      MAGIC_ENABLE
  17      MAGIC_DETECT                  (R)
  18      INTR_ENABLE
  19      INTR_DETECT                   (R)
  20      KEEP_ALIVE_FAIL               (R)
  21      BEACON_FAIL                   (R)
  28-31   CW_BITS
```

ATH:028260h - MAC_PCU_WOW2 ;WOW AIFS/SLOT/TRY_CNT ;hw6

```
  0-7     AIFS
  8-15    SLOT
  16-23   TRY_CNT
```

ATH:028270h - MAC_PCU_WOW3_BEACON_FAIL ;WOW Beacon Fail Enable ;hw6

```
  0       ENABLE
```

ATH:028274h - MAC_PCU_WOW3_BEACON ;WOW Beacon Timeout ;hw6
ATH:028278h - MAC_PCU_WOW3_KEEP_ALIVE ;WOW Keep-Alive Timeout ;hw6

```
  0-31    TIMEOUT
```

ATH:028370h - MAC_PCU_WOW4 ;WOW Pattern Enable/Detect ;hw6
```
  0-7    PATTERN_ENABLE
  8-15   PATTERN_DETECT                (R)
```

ATH:02835Ch - MAC_PCU_WOW5 ;WOW RX Abort Enable ;hw6
```
  0-15   RX_ABORT_ENABLE
```

ATH:02834Ch - MAC_PCU_WOW6 (R) ;WOW RX Buf Start Addr ;hw6
```
  0-15   RXBUF_START_ADDR              (R)
```

ATH:02827Ch - MAC_PCU_WOW_KA ;WOW Auto/Fail/BkoffCs Enable/Disable ;hw6
```
  0      AUTO_DISABLE
  1      FAIL_DISABLE
  2      BKOFF_CS_ENABLE
```

ATH:028294h - PCU_WOW4 ;WOW Offset 0..3 ;hw6
ATH:028298h - PCU_WOW5 ;WOW Offset 4..7 ;hw6
ATH:028378h - PCU_WOW6 ;WOW Offset 8..11 ;hw6
ATH:02837Ch - PCU_WOW7 ;WOW Offset 12..15 ;hw6
```
  0-7    OFFSET0 / OFFSET4 / OFFSET8  / OFFSET12       ;<-- 1st offset in LSBs
  8-15   OFFSET1 / OFFSET5 / OFFSET9  / OFFSET13
  16-23  OFFSET2 / OFFSET6 / OFFSET10 / OFFSET14
  24-31  OFFSET3 / OFFSET7 / OFFSET11 / OFFSET15
```

ATH:028360h - MAC_PCU_WOW_LENGTH1 ;WOW Pattern 0..3 ;hw6
ATH:028364h - MAC_PCU_WOW_LENGTH2 ;WOW Pattern 4..7 ;hw6
ATH:028380h - MAC_PCU_WOW_LENGTH3 ;WOW Pattern 8..11 ;hw6
ATH:028384h - MAC_PCU_WOW_LENGTH4 ;WOW Pattern 12..15 ;hw6
```
  0-7    PATTERN_3 / PATTERN_7 / PATTERN_11 / PATTERN_15
  8-15   PATTERN_2 / PATTERN_6 / PATTERN_10 / PATTERN_14
  16-23  PATTERN_1 / PATTERN_5 / PATTERN_9  / PATTERN_13
  24-31  PATTERN_0 / PATTERN_4 / PATTERN_8  / PATTERN_12   ;1st pattern in MSBs
```

ATH:02828Ch - WOW_EXACT ;WOW Exact Length/Offset 1 ;hw6
ATH:028374h - WOW2_EXACT ;WOW Exact Length/Offset 2 ;hw6
```
  0-7    LENGTH
  8-15   OFFSET
```

ATH:028368h - WOW_PATTERN_MATCH_LESS_THAN_256_BYTES ;hw6
```
  0-15   EN
```

```
 ____________________________________ hw4 only ____________________________________
```

ATH:028020h - PCU_MAX_CFP_DUR ;hw4 only
```
  0-15   VALUE
```
Note: This register does have (almost) the same name as the register below, but without the "MAC_" prefix. In hw6, these two registers have been merged into a single register (called MAC_PCU_MAX_CFP_DUR). In hw2, registers REG_CFP_PERIOD and REG_CFP_DUR might be equivalent?

ATH:028150h - MAC_PCU_MAX_CFP_DUR ;hw4 only
```
  0-3    USEC_FRAC_NUMERATOR
  4-7    USEC_FRAC_DENOMINATOR
```
Note: See "PCU_MAX_CFP_DUR" (other register with similar name, but without "MAC_" prefix).

ATH:0283A4h - MAC_PCU_TSF1_STATUS_L32 ;hw4 only
ATH:0283A8h - MAC_PCU_TSF1_STATUS_U32 ;hw4 only
```
  0-63   VALUE
```

ATH:0283ACh - MAC_PCU_TSF2_STATUS_L32 ;hw4 only
ATH:0283B0h - MAC_PCU_TSF2_STATUS_U32 ;hw4 only
```
  0-63    VALUE
```

ATH:029800h..029FFCh - MAC_PCU_BASEBAND_0[0..511] ;hw4
ATH:02A000h..02BFFCh - MAC_PCU_BASEBAND_1[0..2047] ;hw4
```
  0-31    DATA
```
These two "MAC_PCU" areas are just placeholders for the Baseband Registers at 029800h and up (see BB chapter for details).

ATH:02C000h..02CFFCh - MAC_PCU_BASEBAND_2[0..1023] ;hw4
ATH:02D000h..02DFFCh - MAC_PCU_BASEBAND_3[0..1023] ;hw4
```
  0-31    DATA
```
Unknown what these two "MAC_PCU" areas are intended for.

```
_____________________________________ hw2 only _____________________________________
```

ATH:028500h.. (140h..17Fh) - MAC_PCU_REG_FTYPE[0..3Fh] ;hw2
```
  0       BFCOEF_RX_UPDATE_NORMAL
  1       BFCOEF_RX_UPDATE_SELF_GEN
  2       BFCOEF_TX_ENABLE_NORMAL
  3       BFCOEF_TX_ENABLE_SELF_GEN
  4       BFCOEF_TX_ENABLE_GEN
  5       BFCOEF_TX_ENABLE_MCAST
  6       FILTER_PASS_IF_ALL
  7       FILTER_PASS_IF_DIRECTED
  8       FILTER_PASS_IF_MCAST
  9       FILTER_PASS_IF_BCAST
  10      FILTER_PASS_MC_BC_BSSID
```

ATH:028020h - REG_BEACON ;hw2 only
```
  0-15    BEACON_PERIOD
  16-22   TIM_OFFSET
  23      unspecified
  24      RESET_TSF           <--- related to hw4/hw6: see MAC_PCU_RESET_TSF ?
```

ATH:028024h - REG_CFP_PERIOD ;hw2 only
ATH:028038h - REG_CFP_DUR ;hw2 only
```
  unspecified
```
These two hw2 registers have unspecified content and purpose. Going by the names, they might be similar or equivalent to "MAC_PCU_MAX_CFP_DUR" and "PCU_MAX_CFP_DUR" on hw4 (although if so, unknown which one would be which).

ATH:028028h - REG_TIMER0 ;hw2 only
ATH:02802Ch - REG_TIMER1 ;hw2 only
ATH:028030h - REG_TIMER2 ;hw2 only
ATH:028034h - REG_TIMER3 ;hw2 only
```
  unspecified         ;MAYBE related to MAC_PCU_BT_WL_1..4 or so in hw4/hw6 (?)
```

ATH:02804Ch - REG_TSF_L32 ;hw2 only ... aka MAC_PCU_TSF_L32 ?
ATH:028050h - REG_TSF_U32 ;hw2 only ... aka MAC_PCU_TSF_U32 ?
```
  unspecified
```

ATH:028104h - MAC_PCU_REG_TSF ;hw2 only ;aka MAC_PCU_TSF_ADD_PLL on hw4?
```
  0-7     TSF_INCREMENT          (hw2: ini:1)                    ;-hw2 only
```

ATH:028114h - MAC_PCU_REG_ACKSIFS_RESERVED ;hw2 only

```
  0-7    ACKSIFS_INCREMENT_RESERVED     (hw2: ini:0)           ;-hw2 only
```
Related to the "MAC_PCU_REG_ACKSIFSMEM_RESERVED[0..1Fh]" array?

ATH:028680h.. (1A0h..1BFh) - MAC_PCU_REG_ACKSIFSMEM_RESERVED[0..1Fh] ;hw2
```
  0-9    NORMAL_RESERVED
  10-19  TURBO_RESERVED
```

ATH:028700h.. (1C0h..1DFh) - MAC_PCU_REG_DUR[0..1Fh] ;hw2
```
  0-15   DUR_RATE_TO_DURATION
```

ATH:0287C0h.. (1F0h..1F7h) - MAC_PCU_REG_RTD[0..7] ;hw2
```
  0-4    RTD_RATE_TO_DB_0
  8-12   RTD_RATE_TO_DB_1
  16-20  RTD_RATE_TO_DB_2
  24-28  RTD_RATE_TO_DB_3
```

ATH:0287E0h.. (1F8h..1FFh) - MAC_PCU_REG_DTR[0..7] ;hw2
```
  0-4    DTR_DB_TO_RATE_0
  8-12   DTR_DB_TO_RATE_1
  16-20  DTR_DB_TO_RATE_2
  24-28  DTR_DB_TO_RATE_3
```

ATH:028800h.. (200h..5FFh) - MAC_PCU_REG_KC[0..3FFh] ;hw2
Below bitfields are supposedly somehow stored in multiple words...?
```
  0-31   KC_KEY_31_0               ;aka byte 00h..03h ?
  0-15   KC_KEY_47_32              ;aka byte 04h..05h (and 06h..07h unused?) ?
  0-31   KC_KEY_79_48              ;aka byte 08h..0Bh ?
  0-15   KC_KEY_95_80              ;aka byte 0Ch..0Dh (and 0Eh..0Fh unused?) ?
  0-31   KC_KEY_127_96             ;aka byte 10h..13h ?
  0-2    KC_KEY_TYPE               ;\
  3      KC_LAST_ANTENNA           ;
  4-8    KC_ASYNC_ACK_OFFSET       ;
  9      KC_UPDATE_BEAM_FORMING    ;aka byte 14h..15h (and 16h..17h unused?) ?
  10     KC_RX_CHAIN_0_ACK_ANT     ;
  11     KC_RX_CHAIN_1_ACK_ANT     ;
  12     KC_TX_CHAIN_0_ANT_SEL     ;
  13     KC_TX_CHAIN_1_ANT_SEL     ;
  14     KC_TX_CHAIN_SEL           ;/
  0-31   KC_ADDR_32_1              ;aka byte 18h..1Bh ? (no bit 0 ?)
  0-14   KC_ADDR_47_33             ;aka byte 1Ch..1Dh (and 1Eh..1Fh unused?) ?
  1      KC_VALID                  ;aka byte 20h      (and 21h..xxh unused?) ?
```

ATH:028180h..02819Ch - MAC_PCU_REG_BFCOEF1[0..7] ;hw2
```
  0-23   TSF
  24-30  KEYIDX
  31     KEY_VALID                   (hw2: ini:0)
```

ATH:0281C0h - MAC_PCU_REG_BFCOEF2 ;hw2
```
  0-22   THRESH                      (hw2: ini:0)
  23     unspecified
  24-31  LOCK                        (hw2: ini:0)
```

ATH:0281C4h - MAC_PCU_REG_KCMASK ;hw2
```
  0-15   KCMASK_47_32                (hw2: ini:0000h)
  16     KCMASK_31_0                 (hw2: ini:0)
```

_____ hw2 "MCI" registers _____

Below "MCIxxx" registers exist on hw2 only. Purpose is unknown. There seems to be nothing equivalent in hw4. However, hw6 is having several "MCI_xxx" registers (see WLAN Coex section; unknown if that's related

to hw2 MCI stuff).

ATH:028268h - MAC_PCU_REG_MCICTL ;Control ;hw2

```
  0      MCI_ENABLE              (hw2: ini:0)
  1      OLA_ENABLE              (hw2: ini:1)
  2      PREEMPT_ENABLE          (hw2: ini:1)
  3      CHANNEL_BUSY_ENABLE     (hw2: ini:1)
  4-9    EARLY_NOTIFY_DELAY      (hw2: ini:5)
  10     BMISS_FORCE_WL          (hw2: ini:0)
  11     SLEEP_FORCE_BT          (hw2: ini:1)
  12     HP_QCU_STOMP_BT         (hw2: ini:0)
  31     MCI_BUSY
```

ATH:02826Ch - MAC_PCU_REG_MCIISR ;Interrupt Status ;hw2
ATH:028270h - MAC_PCU_REG_MCIIER ;Interrupt Enable ;hw2

```
  0      ACT_RPT_RCV_INT         (hw2: stat and enable: ini:0)
  1      ACT_DEN_RCV_INT         (hw2: stat and enable: ini:0)
  2      FRQ_RPT_RCV_INT         (hw2: stat and enable: ini:0)
  3      QOS_RPT_RCV_INT         (hw2: stat and enable: ini:0)
  4      GEN_RPT_RCV_INT         (hw2: stat and enable: ini:0)
```

ATH:028274h - MAC_PCU_REG_MCIWLP ;WLP ?? ;hw2 (hw2: ini:0)

```
  unspecified
```

ATH:028278h - MAC_PCU_REG_MCIARW ;AR Write? ;hw2 (hw2: ini:0)
ATH:02827Ch - MAC_PCU_REG_MCIARR ;AR Read? ;hw2
ATH:028280h - MAC_PCU_REG_MCIADW ;AD Write? ;hw2 (hw2: ini:0)
ATH:028284h - MAC_PCU_REG_MCIADR ;AD Read? ;hw2
ATH:028288h - MAC_PCU_REG_MCIFRW ;FR Write? ;hw2 (hw2: ini:0)
ATH:02828Ch - MAC_PCU_REG_MCIFRR ;FR Read? ;hw2
ATH:028290h - MAC_PCU_REG_MCIQRW ;QR Write? ;hw2 (hw2: ini:0)
ATH:028294h - MAC_PCU_REG_MCIQRR ;QR Read? ;hw2
ATH:028298h - MAC_PCU_REG_MCIGRW ;GR Write? ;hw2 (hw2: ini:0)
ATH:02829Ch - MAC_PCU_REG_MCIGRR ;GR Read? ;hw2

```
  unspecified
```

ATH:0282A0h - MAC_PCU_REG_MCISTAT ;Status (counters?) ;hw2

```
  0-7    ACT_RPT_RCV_CNT         (hw2: ini:0)
  8-15   QC_CNT                  (hw2: ini:0)
  16-23  OLA_CNT                 (hw2: ini:0)
  24-31  PREEMPT_CNT             (hw2: ini:0)
```

```
 ___________ hw2 MAC_PCU registers (moved to RTC WLAN in hw4/hw6) ___________
```

ATH:028200h..02821Ch - MAC_PCU_REG_GNRCTMR_N[0..7] ;hw2
ATH:028220h..02823Ch - MAC_PCU_REG_GNRCTMR_P[0..7] ;hw2
ATH:028240h - MAC_PCU_GENERIC_TIMERS_MODE ;aka MAC_PCU_REG_GNRCTMR_M ;hw2
ATH:0280D4h - MAC_PCU_SLP1 ;aka MAC_PCU_REG_SLP1 ;hw2
ATH:0280D8h - MAC_PCU_SLP2 ;aka MAC_PCU_REG_SLP2 ;hw2
ATH:0280DCh - (outcommented) ;aka MAC_PCU_REG_SLP3 ;hw2 (but outcommented)
ATH:028260h - MAC_PCU_SLP3 ;aka old name: MAC_PCU_REG_SLP4 (four) ;hw2
This stuff is located in "WMAC PCU" at 028xxxh in hw2 only. Later versions have it moved to the "RTC WLAN" area (at 004xxxh or 005xxxh), see there for details.

ATH:028244h (mirror of 0040F4h) - MAC_PCU_REG_SLP32_MODE (ini:10F424h) ;hw2
ATH:028248h (mirror of 0040F8h) - MAC_PCU_REG_SLP32_WAKE (ini:07EFh) ;hw2
ATH:02824Ch (mirror of 0040FCh) - MAC_PCU_REG_SLP32_TSF_INC (ini:1E848h) ;hw2

ATH:028250h (mirror of 004100h) - MAC_PCU_REG_SLPMIB1 ;hw2
ATH:028254h (mirror of 004104h) - MAC_PCU_REG_SLPMIB2 ;hw2
ATH:028258h (mirror of 004108h) - MAC_PCU_REG_SLPMIB3 ;hw2
ATH:028264h (mirror of 00410Ch) - MAC_PCU_REG_SLP5 (ini: 0FFFFFFh) ;hw2
These hw2 registers seem to be just mirrors of other hw2 registers in the RTC area at 004xxxh (see RTC WLAN chapter for details).
In hw4/hw6, the mirrors at 028xxxh are removed (and only the registers at 004xxxh are kept; whereas, in hw6 that part moved to 005xxxh).

_____ outcommented hw2 stuff _____

Below outcommented stuff is found in hw2 source code, maybe it was used in older hw2 revisions (in case there multiple hw2 revisions), or maybe it was used in even older pre-hw2 chips, or maybe it's just some experimental stuff that was never implemented in hardware.

ATH:028140h - outcommented:MAC_PCU_REG_TSFCAL ;Misc ;hw2
```
0-3    outcommented:COUNT          (hw2: ini:8)       ;\
4-7    outcommented:INTERVAL       (hw2: ini:0Ah)     ; hw2 only
8      outcommented:ENABLE         (hw2: ini:1)       ; (although it's
9      outcommented:AUTO_CAL       (hw2: ini:1)       ; outcommented even
10     outcommented:VALUE_WE       (hw2: ini:0)       ; in hw2 source code)
16-31  outcommented:VALUE          (hw2: ini:8000h)   ;/
```

ATH:02814Ch - outcommented:MAC_PCU_REG_SYNC2 ;Misc ;hw2
```
0-7    outcommented:TIME_OFFSET    (hw2: ini:0)
8      outcommented:MASTER         (hw2: ini:0)
9      outcommented:REPLACE        (hw2: ini:0)
10     outcommented:TUNE           (hw2: ini:0)
11     outcommented:CLEAR          (hw2: ini:0)
16-31  outcommented:INTR_THRESH    (hw2: ini:FFFFh)
```

ATH:028148h - outcommented:MAC_PCU_REG_SYNC1 ;Time (ini:0) ;hw2
ATH:028158h - outcommented:MAC_PCU_REG_SYNC5 ;RX Time ;hw2
ATH:028160h - outcommented:MAC_PCU_REG_SYNC7 ;Last Time ;hw2
ATH:028164h - outcommented:MAC_PCU_REG_SYNC8 ;Updated Time ;hw2
```
0-30   outcommented:TIME
31     -
```

ATH:028150h - outcommented:MAC_PCU_REG_SYNC3 ;MCAST Addr_L ;hw2
ATH:028154h - outcommented:MAC_PCU_REG_SYNC4 ;MCAST Addr_U ;hw2
```
0-47   outcommented:MCAST_ADDR     (hw2: ini:0)
48-63  -
```

ATH:02815Ch - outcommented:MAC_PCU_REG_SYNC6 ;INC ;hw2
```
0-31   outcommented:INC
```

DSi Atheros Wifi - Internal I/O - 029800h - BB Baseband (hw4/hw6)

ATH:029800h/02A360h - BB_TEST_CONTROLS ;hw4/hw6
```
0-3    CF_TSTTRIG_SEL
4      CF_TSTTRIG
5-6    CF_RFSHIFT_SEL
8-9    CARDBUS_MODE
10     CLKOUT_IS_CLK32
13     ENABLE_RFSILENT_BB
15     ENABLE_MINI_OBS
17     SLOW_CLK160
```

```
18      AGC_OBS_SEL_3
19-22   CF_BBB_OBS_SEL
23      RX_OBS_SEL_5TH_BIT
24      AGC_OBS_SEL_4
28      FORCE_AGC_CLEAR
30-31   TSTDAC_OUT_SEL
```

ATH:029804h/02A204h - BB_GEN_CONTROLS ;hw4/hw6

```
0       TURBO
1       CF_SHORT20
2       DYN_20_40
3       DYN_20_40_PRI_ONLY
4       DYN_20_40_PRI_CHN
5       DYN_20_40_EXT_CHN
6       HT_ENABLE
7       ALLOW_SHORT_GI
8       CF_2_CHAINS_USE_WALSH
9       hw4: CF_SINGLE_HT_LTF1                          ;-hw4
9       hw6: CF_3_CHAINS_USE_WALSH                      ;-hw6
10      GF_ENABLE
11      hw4: BYPASS_DAC_FIFO_N                          ;-hw4
11      hw6: ENABLE_DAC_ASYNC_FIFO                      ;\
14      hw6: BOND_OPT_CHAIN_SEL                         ;
15      hw6: STATIC20_MODE_HT40_PACKET_HANDLING         ;
16      hw6: STATIC20_MODE_HT40_PACKET_ERROR_RPT        ; hw6
17      hw6: ENABLE_CSD_PHASE_DITHERING                 ;
18-24   hw6: UNSUPP_HT_RATE_THRESHOLD                   ;
25      hw6: EN_ERR_TX_CHAIN_MASK_ZERO                  ;
26      hw6: IS_MCKINLEY_TPC                            ;/
```

ATH:029808h/02A364h - BB_TEST_CONTROLS_STATUS ;hw4/hw6

```
0       CF_TSTDAC_EN
1       CF_TX_SRC_IS_TSTDAC
2-4     CF_TX_OBS_SEL
5-6     CF_TX_OBS_MUX_SEL
7       CF_TX_SRC_ALTERNATE
8       CF_TSTADC_EN
9       CF_RX_SRC_IS_TSTADC
10-13   RX_OBS_SEL
14      DISABLE_A2_WARM_RESET
15      RESET_A2
16-18   AGC_OBS_SEL
19      CF_ENABLE_FFT_DUMP
23      CF_DEBUGPORT_IN
27      DISABLE_AGC_TO_A2
28      CF_DEBUGPORT_EN
29-30   CF_DEBUGPORT_SEL
```

ATH:02980Ch/029800h - BB_TIMING_CONTROLS_1 ;hw4/hw6

```
0-6     STE_THR
7-12    STE_TO_LONG1
13-16   TIMING_BACKOFF
17      ENABLE_HT_FINE_PPM
18-19   HT_FINE_PPM_STREAM
20-21   HT_FINE_PPM_QAM
22      ENABLE_LONG_CHANFIL
23      ENABLE_RX_STBC
24      ENABLE_CHANNEL_FILTER
25-26   FALSE_ALARM
27      ENABLE_LONG_RESCALE
28      TIMING_LEAK_ENABLE
29-30   COARSE_PPM_SELECT
31      FFT_SCALING
```

ATH:029810h/029804h - BB_TIMING_CONTROLS_2 ;hw4/hw6

0-11	FORCED_DELTA_PHI_SYMBOL
12	FORCE_DELTA_PHI_SYMBOL
13	ENABLE_MAGNITUDE_TRACK
14	ENABLE_SLOPE_FILTER
15	ENABLE_OFFSET_FILTER
16-22	DC_OFF_DELTAF_THRES
24-26	DC_OFF_TIM_CONST
27	ENABLE_DC_OFFSET
28	ENABLE_DC_OFFSET_TRACK
29	ENABLE_WEIGHTING
30	TRACEBACK128
31	ENABLE_HT_FINE_TIMING

ATH:029814h/029808h - BB_TIMING_CONTROLS_3 ;hw4/hw6

0-7	PPM_RESCUE_INTERVAL
8	ENABLE_PPM_RESCUE
9	ENABLE_FINE_PPM
10	ENABLE_FINE_INTERP
11	CONTINUOUS_PPM_RESCUE
12	ENABLE_DF_CHANEST
13-16	DELTA_SLOPE_COEF_EXP
17-31	DELTA_SLOPE_COEF_MAN

ATH:029818h/02A200h - BB_D2_CHIP_ID (R) ;hw4/hw6

0-7	OLD_ID	(R)
8-31	ID	(R)

ATH:02981Ch/02A20Ch - BB_ACTIVE ;hw4/hw6

0	CF_ACTIVE

ATH:029820h/02A258h - BB_TX_TIMING_1 ;hw4/hw6

0-7	TX_FRAME_TO_ADC_OFF
8-15	TX_FRAME_TO_A2_RX_OFF
16-23	TX_FRAME_TO_DAC_ON
24-31	TX_FRAME_TO_A2_TX_ON

ATH:029824h/02A25Ch - BB_TX_TIMING_2 ;hw4/hw6

0-7	TX_FRAME_TO_TX_D_START
8-15	TX_FRAME_TO_PA_ON
16-23	TX_END_TO_PA_OFF
24-31	TX_END_TO_A2_TX_OFF

ATH:029828h/02A260h - BB_TX_TIMING_3 ;hw4/hw6

0-7	TX_END_TO_DAC_OFF
8-15	TX_FRAME_TO_THERM_CHAIN_ON
16-23	TX_END_TO_A2_RX_ON
24-31	TX_END_TO_ADC_ON

ATH:02982Ch/02A350h - BB_ADDAC_PARALLEL_CONTROL ;hw4/hw6

12	OFF_DACLPMODE
13	OFF_PWDDAC
15	OFF_PWDADC
28	ON_DACLPMODE
29	ON_PWDDAC
31	ON_PWDADC

ATH:029834h/02A264h - BB_XPA_TIMING_CONTROL ;hw4/hw6

0-7	TX_FRAME_TO_XPAA_ON
8-15	TX_FRAME_TO_XPAB_ON

```
  16-23  TX_END_TO_XPAA_OFF
  24-31  TX_END_TO_XPAB_OFF
```

ATH:029838h/02A280h - BB_MISC_PA_CONTROL ;hw4/hw6

```
  0      XPAA_ACTIVE_HIGH
  1      XPAB_ACTIVE_HIGH
  2      ENABLE_XPAA
  3      ENABLE_XPAB
```

ATH:02983Ch/0298A4h - BB_TSTDAC_CONSTANT ;hw4/hw6

```
  0-10   CF_TSTDAC_CONSTANT_I
  11-21  CF_TSTDAC_CONSTANT_Q
```

ATH:029840h/029820h - BB_FIND_SIGNAL_LOW ;hw4/hw6

```
  0-5    RELSTEP_LOW
  6-11   FIRSTEP_LOW
  12-19  FIRPWR_LOW
  20-23  YCOK_MAX_LOW
  24-30  LONG_SC_THRESH
```

ATH:029844h/029E00h - BB_SETTLING_TIME ;hw4/hw6

```
  0-6    AGC_SETTLING
  7-13   SWITCH_SETTLING
  14-19  ADCSAT_THRL
  20-25  ADCSAT_THRH
  26-29  LBRESET_ADVANCE
```

ATH:029848h/029E04h - BB_GAIN_FORCE_MAX_GAINS_B0 ;hw4/hw6
ATH:02A848h/02AE04h - BB_GAIN_FORCE_MAX_GAINS_B1 ;hw4/hw6

```
  7-13   hw4: XATTEN1_HYST_MARGIN_0/1   ;\      ;\separate settings in B0/B1
  14-20  hw4: XATTEN2_HYST_MARGIN_0/1   ; hw4   ;/
  21     hw4: GAIN_FORCE                ;       ;\global setting (not in B1)
  31     hw4: ENABLE_SHARED_RX          ;/      ;/
  0-7    hw6: RF_GAIN_F_0/1             ;\      ;\
  8-15   hw6: MB_GAIN_F_0/1             ;       ;
  16     hw6: XATTEN1_SW_F_0/1          ; hw6   ; separate settings in B0/B1
  17     hw6: XATTEN2_SW_F_0/1          ;       ;
  18-24  hw6: XATTEN1_HYST_MARGIN_0/1   ;       ;
  25-31  hw6: XATTEN2_HYST_MARGIN_0/1   ;/      ;/
```

ATH:02984Ch - BB_GAINS_MIN_OFFSETS_B0 ;hw4
ATH:02A84Ch - BB_GAINS_MIN_OFFSETS_B1 ;hw4
ATH:029E08h - BB_GAINS_MIN_OFFSETS ;hw6 (only global setting for B0 and B1)

```
  0-6    OFFSETC1                       ;\             ;\global setting
  7-11   OFFSETC2                       ; hw4/hw6      ; (not in B1 register)
  12-16  OFFSETC3                       ;/             ;/
  17-24  hw4: RF_GAIN_F_0/1             ;\             ;\separate settings
  25     hw4: XATTEN1_SW_F_0/1          ; hw4          ; in B0/B1 registers
  26     hw4: XATTEN2_SW_F_0/1          ;/             ;/
  17     hw6: GAIN_FORCE                ;\             ;\
  18     hw6: CF_AGC_HIST_ENABLE        ;              ; global setting
  19     hw6: CF_AGC_HIST_GC            ; hw6          ; (hw6 doesn't have
  20     hw6: CF_AGC_HIST_VOTING        ;              ; a B1 register at all)
  21     hw6: CF_AGC_HIST_PHY_ERR       ;/             ;/
```

ATH:029850h/029E0Ch - BB_DESIRED_SIGSIZE ;hw4/hw6

```
  0-7    ADC_DESIRED_SIZE
  20-27  TOTAL_DESIRED
  28-29  INIT_GC_COUNT_MAX
  30     REDUCE_INIT_GC_COUNT
  31     ENA_INIT_GAIN
```

ATH:029854h/029C00h - BB_TIMING_CONTROL_3A ;hw4/hw6

```
0-6    STE_THR_HI_RSSI                    ;-hw4/hw6
7      hw6: USE_HTSIG1_20_40_BW_VALUE     ;-hw6
```

ATH:029858h/029E10h - BB_FIND_SIGNAL ;hw4/hw6

```
0-5    RELSTEP
6-11   RELPWR
12-17  FIRSTEP
18-25  FIRPWR
26-31  M1COUNT_MAX
```

ATH:02985Ch/029E14h - BB_AGC ;hw4/hw6

```
0-6    COARSEPWR_CONST
7-14   COARSE_LOW
15-21  COARSE_HIGH
22-29  QUICK_DROP
30-31  RSSI_OUT_SELECT
```

ATH:029860h/02A2C4h - BB_AGC_CONTROL ;hw4/hw6

```
0      DO_CALIBRATE
1      DO_NOISEFLOOR
3-5    MIN_NUM_GAIN_CHANGE
6-9    YCOK_MAX
10     LEAKY_BUCKET_ENABLE
11     CAL_ENABLE
12     USE_TABLE_SEED
13     AGC_UPDATE_TABLE_SEED
15     ENABLE_NOISEFLOOR
16     ENABLE_FLTR_CAL
17     NO_UPDATE_NOISEFLOOR
18     EXTEND_NF_PWR_MEAS
19     CLC_SUCCESS                (R)
20     ENABLE_PKDET_CAL
```

ATH:029864h/029E1Ch - BB_CCA_B0 ;hw4/hw6
ATH:02AE1Ch - BB_CCA_B1 ;hw6

```
0-8    CF_MAXCCAPWR_0                     ;-separate settings (on hw6)
9-11   CF_CCA_COUNT_MAXC                  ;\global setting (not in B1)
12-19  CF_THRESH62                        ;/
20-28  MINCCAPWR_0                   (R)  ;-separate settings (on hw6)
```

ATH:029868h/029824h - BB_SFCORR ;hw4/hw6

```
0-4    M2COUNT_THR
5-10   ADCSAT_THRESH
11-16  ADCSAT_ICOUNT
17-23  M1_THRES
24-30  M2_THRES
```

ATH:02986Ch/029828h - BB_SELF_CORR_LOW ;hw4/hw6

```
0      USE_SELF_CORR_LOW
1-7    M1COUNT_MAX_LOW
8-13   M2COUNT_THR_LOW
14-20  M1_THRESH_LOW
21-27  M2_THRESH_LOW
```

ATH:029874h/02A340h - BB_SYNTH_CONTROL ;hw4/hw6

```
0-16   RFCHANFRAC
17-25  RFCHANNEL
26-27  RFAMODEREFSEL
28     RFFRACMODE
```

```
29      RFBMODE
30      RFSYNTH_CTRL_SSHIFT
```

ATH:029878h/02A344h - BB_ADDAC_CLK_SELECT ;hw4/hw6

```
2-3     BB_DAC_CLK_SELECT
4-5     BB_ADC_CLK_SELECT
```

ATH:02987Ch/02A348h - BB_PLL_CNTL ;hw4/hw6

```
0-9     BB_PLL_DIV
10-13   BB_PLL_REFDIV
14-15   BB_PLL_CLK_SEL
16      BB_PLLBYPASS
17-27   BB_PLL_SETTLE_TIME
```

ATH:029900h/02A220h - BB_VIT_SPUR_MASK_A ;CF_PUNC_MASK_A ;hw4/hw6
ATH:029904h/02A224h - BB_VIT_SPUR_MASK_B ;CF_PUNC_MASK_B ;hw4/hw6

```
0-9     CF_PUNC_MASK_A     / CF_PUNC_MASK_B
10-16   CF_PUNC_MASK_IDX_A / CF_PUNC_MASK_IDX_B
```

ATH:029908h/029C0Ch - BB_PILOT_SPUR_MASK ;CF_PILOT_MASK_A/B ;hw4/hw6
ATH:02990Ch/029C10h - BB_CHAN_SPUR_MASK ;CF_CHAN_MASK_A/B ;hw4/hw6

```
0-4     CF_PILOT_MASK_A     / CF_CHAN_MASK_A
5-11    CF_PILOT_MASK_IDX_A / CF_CHAN_MASK_IDX_A
12-16   CF_PILOT_MASK_B     / CF_CHAN_MASK_B
17-23   CF_PILOT_MASK_IDX_B / CF_CHAN_MASK_IDX_B
```

ATH:029910h/02A228h - BB_SPECTRAL_SCAN ;hw4/hw6

```
0       SPECTRAL_SCAN_ENA
1       SPECTRAL_SCAN_ACTIVE
2       DISABLE_RADAR_TCTL_RST
3       DISABLE_PULSE_COARSE_LOW
4-7     SPECTRAL_SCAN_FFT_PERIOD
8-15    SPECTRAL_SCAN_PERIOD
16-27   SPECTRAL_SCAN_COUNT
28      SPECTRAL_SCAN_SHORT_RPT
29      SPECTRAL_SCAN_PRIORITY
30      SPECTRAL_SCAN_USE_ERR5
31      hw6: SPECTRAL_SCAN_COMPRESSED_RPT       ;-hw6
```

ATH:02A248h - BB_SPECTRAL_SCAN_2 ;hw6

```
0       hw6: SPECTRAL_SCAN_RPT_MODE             ;\hw6
1-8     hw6: SPECTRAL_SCAN_NOISE_FLOOR_REF      ;/
```

ATH:029914h/02A254h - BB_ANALOG_POWER_ON_TIME ;hw4/hw6

```
0-13    ACTIVE_TO_RECEIVE
```

ATH:029918h/02A230h - BB_SEARCH_START_DELAY ;hw4/hw6

```
0-11    SEARCH_START_DELAY
12      ENABLE_FLT_SVD
13      ENABLE_SEND_CHAN
14      hw6: RX_SOUNDING_ENABLE
15      hw6: RM_HCSD4SVD
```

ATH:02991Ch/02A234h - BB_MAX_RX_LENGTH ;hw4/hw6

```
0-11    MAX_RX_LENGTH
12-29   MAX_HT_LENGTH
```

ATH:029920h/02980Ch - BB_TIMING_CONTROL_4 ;hw4/hw6

```
12-15   CAL_LG_COUNT_MAX
16      DO_GAIN_DC_IQ_CAL
```

17-20 USE_PILOT_TRACK_DF
21-27 EARLY_TRIGGER_THR
28 ENABLE_PILOT_MASK
29 ENABLE_CHAN_MASK
30 ENABLE_SPUR_FILTER
31 ENABLE_SPUR_RSSI

ATH:029924h/029810h - BB_TIMING_CONTROL_5 ;hw4/hw6

0 ENABLE_CYCPWR_THR1
1-7 CYCPWR_THR1
15 ENABLE_RSSI_THR1A
16-22 RSSI_THR1A
23-29 LONG_SC_THRESH_HI_RSSI
30 FORCED_AGC_STR_PRI
31 FORCED_AGC_STR_PRI_EN

ATH:029928h/02A7D0h - BB_PHYONLY_WARM_RESET ;hw4/hw6

0 PHYONLY_RST_WARM_L

ATH:02992Ch/02A7D4h - BB_PHYONLY_CONTROL ;hw4/hw6

0 RX_DRAIN_RATE
1 LATE_TX_SIGNAL_SYMBOL
2 GENERATE_SCRAMBLER
3 TX_ANTENNA_SELECT
4 STATIC_TX_ANTENNA
5 RX_ANTENNA_SELECT
6 STATIC_RX_ANTENNA
7 EN_LOW_FREQ_SLEEP

ATH:02993Ch/02A3F0h - BB_POWERTX_MAX ;hw4/hw6

6 USE_PER_PACKET_POWERTX_MAX
7 hw6: USE_PER_PACKET_OLPC_GAIN_DELTA_ADJ

ATH:029940h/02983Ch - BB_EXTENSION_RADAR ;hw4/hw6

8-13 BLOCKER40_MAX_RADAR
14 ENABLE_EXT_RADAR
15-22 RADAR_DC_PWR_THRESH
23-30 RADAR_LB_DC_CAP
31 DISABLE_ADCSAT_HOLD

ATH:029944h/02A238h - BB_FRAME_CONTROL ;hw4/hw6

0-1 CF_OVERLAP_WINDOW
2 CF_SCALE_SHORT
3-5 CF_TX_CLIP
6-7 CF_TX_DOUBLESAMP_DAC
8-15 TX_END_ADJUST
16 PREPEND_CHAN_INFO
17 SHORT_HIGH_PAR_NORM
18 EN_ERR_GREEN_FIELD
19 hw4: EN_ERR_XR_POWER_RATIO
19 hw6: EN_ERR_STATIC20_MODE_HT40_PACKET
20 EN_ERR_OFDM_XCORR
21 EN_ERR_LONG_SC_THR
22 EN_ERR_TIM_LONG1
23 EN_ERR_TIM_EARLY_TRIG
24 EN_ERR_TIM_TIMEOUT
25 EN_ERR_SIGNAL_PARITY
26 EN_ERR_RATE_ILLEGAL
27 EN_ERR_LENGTH_ILLEGAL
28 hw4: EN_ERR_HT_SERVICE
28 hw6: NO_6MBPS_SERVICE_ERR
29 EN_ERR_SERVICE

```
  30      EN_ERR_TX_UNDERRUN
  31      EN_ERR_RX_ABORT
```

ATH:029948h/029814h - BB_TIMING_CONTROL_6 ;hw4/hw6

```
  0-7     HI_RSSI_THRESH
  8-14    EARLY_TRIGGER_THR_HI_RSSI
  15-20   OFDM_XCORR_THRESH
  21-27   OFDM_XCORR_THRESH_HI_RSSI
  28-31   LONG_MEDIUM_RATIO_THR
```

ATH:02994Ch/02981Ch - BB_SPUR_MASK_CONTROLS ;hw4/hw6

```
  0-7     SPUR_RSSI_THRESH
  8       EN_VIT_SPUR_RSSI
  17      ENABLE_MASK_PPM
  18-25   MASK_RATE_CNTL
  26      hw6: ENABLE_NF_RSSI_SPUR_MIT
```

ATH:029950h/0298DCh - BB_RX_IQ_CORR_B0 ;hw4/hw6
ATH:02A8DCh - BB_RX_IQ_CORR_B1 ;hw6

```
  0-6     RX_IQCORR_Q_Q_COFF_0/1          ;\separate settings (on hw6)
  7-13    RX_IQCORR_Q_I_COFF_0/1          ;/
  14      RX_IQCORR_ENABLE                ;-global setting (not in B1)
  15-21   LOOPBACK_IQCORR_Q_Q_COFF_0/1    ;\separate settings (on hw6)
  22-28   LOOPBACK_IQCORR_Q_I_COFF_0/1    ;/
  29      LOOPBACK_IQCORR_ENABLE          ;-global setting (not in B1)
```

ATH:029954h/029834h - BB_RADAR_DETECTION ;hw4/hw6

```
  0       PULSE_DETECT_ENABLE
  1-5     PULSE_IN_BAND_THRESH
  6-11    PULSE_RSSI_THRESH
  12-17   PULSE_HEIGHT_THRESH
  18-23   RADAR_RSSI_THRESH
  24-30   RADAR_FIRPWR_THRESH
  31      ENABLE_RADAR_FFT
```

ATH:029958h/029838h - BB_RADAR_DETECTION_2 ;hw4/hw6

```
  0-7     RADAR_LENGTH_MAX
  8-12    PULSE_RELSTEP_THRESH
  13      ENABLE_PULSE_RELSTEP_CHECK
  14      ENABLE_MAX_RADAR_RSSI
  15      ENABLE_BLOCK_RADAR_CHECK
  16-21   RADAR_RELPWR_THRESH
  22      RADAR_USE_FIRPWR_128
  23      ENABLE_RADAR_RELPWR_CHECK
  24-26   CF_RADAR_BIN_THRESH_SEL
  27      ENABLE_PULSE_GC_COUNT_CHECK
```

ATH:02995Ch/0298D0h - BB_TX_PHASE_RAMP_B0 ;hw4/hw6
ATH:02A8D0h - BB_TX_PHASE_RAMP_B1 ;hw6

```
  0       CF_PHASE_RAMP_ENABLE
  1-6     CF_PHASE_RAMP_BIAS
  7-16    CF_PHASE_RAMP_INIT
  17-24   CF_PHASE_RAMP_ALPHA
```

ATH:029960h/02A284h - BB_SWITCH_TABLE_CHN_B0 ;hw4/hw6
ATH:02B284h - BB_SWITCH_TABLE_CHN_B1 ;hw6

```
  0-1     SWITCH_TABLE_IDLE
  2-3     SWITCH_TABLE_T
  4-5     SWITCH_TABLE_R
  6-7     SWITCH_TABLE_RX1
```

```
  8-9    SWITCH_TABLE_RX12
  10-11  SWITCH_TABLE_B
```

ATH:029964h/02A288h - BB_SWITCH_TABLE_COM1 ;hw4/hw6

```
  0-3    SWITCH_TABLE_COM_IDLE
  4-7    SWITCH_TABLE_COM_T1
  8-11   SWITCH_TABLE_COM_T2
  12-15  SWITCH_TABLE_COM_B
  16-19  hw6: SWITCH_TABLE_COM_IDLE_ALT         ;\hw6
  20-23  hw6: SWITCH_TABLE_COM_SPDT             ;/
```

ATH:029968h/029E20h - BB_CCA_CTRL_2_B0 ;hw4/hw6
ATH:02AE20h - BB_CCA_CTRL_2_B1 ;hw6

```
  0-8    MINCCAPWR_THR_0/1              ;-separate settings (on hw6)
  9      ENABLE_MINCCAPWR_THR           ;-global setting (not in B1)
  10-17  NF_GAIN_COMP_0/1               ;-separate settings (on hw6)
  18     THRESH62_MODE                  ;-global setting (not in B1)
```

ATH:02996Ch/02A28Ch - BB_SWITCH_TABLE_COM2 ;hw4/hw6

```
  0-3    hw4: SWITCH_TABLE_COM_RA1NXAL1         ;\
  4-7    hw4: SWITCH_TABLE_COM_RA2NXAL1         ;
  8-11   hw4: SWITCH_TABLE_COM_RA1XAL1          ;
  12-15  hw4: SWITCH_TABLE_COM_RA2XAL1          ; hw4
  16-19  hw4: SWITCH_TABLE_COM_RA1NXAL2         ;
  20-23  hw4: SWITCH_TABLE_COM_RA2NXAL2         ;
  24-27  hw4: SWITCH_TABLE_COM_RA1XAL2          ;
  28-31  hw4: SWITCH_TABLE_COM_RA2XAL2          ;/
  0-3    hw6: SWITCH_TABLE_COM_RA1L1            ;\
  4-7    hw6: SWITCH_TABLE_COM_RA2L1            ;
  8-11   hw6: SWITCH_TABLE_COM_RA1L2            ; hw6
  12-15  hw6: SWITCH_TABLE_COM_RA2L2            ;
  16-19  hw6: SWITCH_TABLE_COM_RA12             ;/
```

ATH:029970h/029E24h - BB_RESTART ;hw4/hw6

```
  0      ENABLE_RESTART
  1-5    RESTART_LGFIRPWR_DELTA
  6      ENABLE_PWR_DROP_ERR
  7-11   PWRDROP_LGFIRPWR_DELTA
  12-17  OFDM_CCK_RSSI_BIAS
  18-20  ANT_FAST_DIV_GC_LIMIT
  21     ENABLE_ANT_FAST_DIV_M2FLAG
  22-28  WEAK_RSSI_VOTE_THR
  29     ENABLE_PWR_DROP_ERR_CCK
  30     DISABLE_DC_RESTART
  31     RESTART_MODE_BW40
```

ATH:029978h/02A390h - BB_SCRAMBLER_SEED ;hw4/hw6

```
  0-6    FIXED_SCRAMBLER_SEED
```

ATH:02997Ch/02A23Ch - BB_RFBUS_REQUEST ;hw4/hw6

```
  0      RFBUS_REQUEST
```

ATH:0299A0h/029818h - BB_TIMING_CONTROL_11 ;hw4/hw6

```
  0-19   SPUR_DELTA_PHASE
  20-29  SPUR_FREQ_SD
  30     USE_SPUR_FILTER_IN_AGC
  31     USE_SPUR_FILTER_IN_SELFCOR
```

ATH:0299A4h/02A2A0h - BB_MULTICHAIN_ENABLE ;hw4/hw6

```
  0-2    RX_CHAIN_MASK
```

ATH:0299A8h/029880h - BB_MULTICHAIN_CONTROL ;hw4/hw6

```
0      FORCE_ANALOG_GAIN_DIFF
1-7    FORCED_GAIN_DIFF_01
8      SYNC_SYNTHON
9      USE_POSEDGE_REFCLK
10-20  CF_SHORT_SAT
22-28  FORCED_GAIN_DIFF_02
29     FORCE_SIGMA_ZERO
```

ATH:0299ACh/029E28h - BB_MULTICHAIN_GAIN_CTRL ;hw4/hw6

```
0-7    QUICKDROP_LOW
8      ENABLE_CHECK_STRONG_ANT
9-14   ANT_FAST_DIV_BIAS
15-20  CAP_GAIN_RATIO_SNR
21     CAP_GAIN_RATIO_ENA
22     CAP_GAIN_RATIO_MODE
23     ENABLE_ANT_SW_RX_PROT
24     ENABLE_ANT_DIV_LNADIV
25-26  ANT_DIV_ALT_LNACONF
27-28  ANT_DIV_MAIN_LNACONF
29     ANT_DIV_ALT_GAINTB
30     ANT_DIV_MAIN_GAINTB
```

ATH:0299B4h/0298D4h - BB_ADC_GAIN_DC_CORR_B0 ;hw4/hw6
ATH:02A8D4h - BB_ADC_GAIN_DC_CORR_B1 ;hw6

```
0-5    ADC_GAIN_CORR_Q_COEFF_0/1      ;\
6-11   ADC_GAIN_CORR_I_COEFF_0/1      ; separate settings (on hw6)
12-20  ADC_DC_CORR_Q_COEFF_0/1        ;
21-29  ADC_DC_CORR_I_COEFF_0/1        ;/
30     ADC_GAIN_CORR_ENABLE           ;\global setting (not in B1)
31     ADC_DC_CORR_ENABLE             ;/
```

ATH:0299B8h/029E2Ch - BB_EXT_CHAN_PWR_THR_1 ;hw4/hw6

```
0-7    THRESH62_EXT
8-15   ANT_DIV_ALT_ANT_MINGAINIDX
16-20  ANT_DIV_ALT_ANT_DELTAGAINIDX
21-26  ANT_DIV_ALT_ANT_DELTANF
```

ATH:0299BCh/029830h - BB_EXT_CHAN_PWR_THR_2_B0 ;hw4/hw6
ATH:02A830h - BB_EXT_CHAN_PWR_THR_2_B1 ;hw6

```
0-8    CF_MAXCCAPWR_EXT_0/1           ;-separate settings (on hw6)
9-15   CYCPWR_THR1_EXT                ;-global setting (not in B1)
16-24  MINCCAPWR_EXT_0/1         (R)  ;-separate settings (on hw6)
```

ATH:0299C0h/02982Ch - BB_EXT_CHAN_SCORR_THR ;hw4/hw6

```
0-6    M1_THRES_EXT
7-13   M2_THRES_EXT
14-20  M1_THRES_LOW_EXT
21-27  M2_THRES_LOW_EXT
28     SPUR_SUBCHANNEL_SD
```

ATH:0299C4h/029E30h - BB_EXT_CHAN_DETECT_WIN ;hw4/hw6

```
0-3    DET_DIFF_WIN_WEAK
4-7    DET_DIFF_WIN_WEAK_LOW
8-12   DET_DIFF_WIN_WEAK_CCK
13-15  DET_20H_COUNT
16-18  DET_EXT_BLK_COUNT
19-24  WEAK_SIG_THR_CCK_EXT
25-28  DET_DIFF_WIN_THRESH
```

ATH:0299C8h/029E34h - BB_PWR_THR_20_40_DET ;hw4/hw6

```
0-4    PWRDIFF40_THRSTR
5-10   BLOCKER40_MAX
11-15  DET40_PWRSTEP_MAX
16-23  DET40_THR_SNR
24-28  DET40_PRI_BIAS
29     PWRSTEP40_ENA
30     LOWSNR40_ENA
```

ATH:0299D0h/029C14h - BB_SHORT_GI_DELTA_SLOPE ;hw4/hw6

```
0-3    DELTA_SLOPE_COEF_EXP_SHORT_GI
4-18   DELTA_SLOPE_COEF_MAN_SHORT_GI
```

ATH:0299DCh/02A370h - BB_CHANINFO_CTRL ;hw4/hw6

```
0      CAPTURE_CHAN_INFO
1      DISABLE_CHANINFOMEM
2      hw6: CAPTURE_SOUNDING_PACKET
3      hw6: CHANINFOMEM_S2_READ
```

ATH:0299E0h/02A3A4h - BB_HEAVY_CLIP_CTRL ;hw4/hw6

```
0-8    CF_HEAVY_CLIP_ENABLE
9      PRE_EMP_HT40_ENABLE
10-17  hw6: HEAVY_CLIP_FACTOR_XR   ;-hw6 (moved from hw4's BB_RIFS_SRCH)
```

ATH:0299E4h/02A3A8h - BB_HEAVY_CLIP_20 ;FACTOR_0..3 ;hw4/hw6
ATH:0299E8h/02A3ACh - BB_HEAVY_CLIP_40 ;FACTOR_4..7 ;hw4/hw6

```
0-7    HEAVY_CLIP FACTOR_0 / FACTOR_4
8-15   HEAVY_CLIP FACTOR_1 / FACTOR_5
16-23  HEAVY_CLIP FACTOR_2 / FACTOR_6
24-31  HEAVY_CLIP FACTOR_3 / FACTOR_7
```

ATH:0299ECh/029E38h - BB_RIFS_SRCH ;hw4/hw6

```
0-7    hw4: HEAVY_CLIP_FACTOR_XR   ;-hw4 (moved to BB_HEAVY_CLIP_CTRL in hw6)
8-15   INIT_GAIN_DB_OFFSET
16-25  RIFS_INIT_DELAY
26     RIFS_DISABLE_PWRLOW_GC
27     RIFS_DISABLE_CCK_DET
```

ATH:0299F0h/02A2C8h - BB_IQ_ADC_CAL_MODE ;hw4/hw6

```
0-1    GAIN_DC_IQ_CAL_MODE
2      TEST_CALADCOFF
```

ATH:0299FCh/029884h - BB_PER_CHAIN_CSD ;hw4/hw6

```
0-4    CSD_CHN1_2CHAINS
5-9    CSD_CHN1_3CHAINS
10-14  CSD_CHN2_3CHAINS
```

ATH:029A00h..029BFCh - BB_RX_OCGAIN[0..127] (W) ;hw4
ATH:02A000h..02A1FCh - BB_RX_OCGAIN[0..127] (R/W?) ;hw6
ATH:02AA00h..02ABFCh - BB_RX_OCGAIN2[0..127] (W) ;hw4
ATH:02B000h..02B1FCh - BB_RX_OCGAIN2[0..127] (R/W?) ;hw6

```
0-31   GAIN_ENTRY
```

ATH:029C00h/0298A0h - BB_TX_CRC (R) ;hw4/hw6

```
0-15   TX_CRC                          (R)
```

ATH:029C10h/0298C0h - BB_IQ_ADC_MEAS_0_B0 (R) ;hw4/hw6
ATH:029C14h/0298C4h - BB_IQ_ADC_MEAS_1_B0 (R) ;hw4/hw6

ATH:029C18h/0298C8h - BB_IQ_ADC_MEAS_2_B0 (R) ;hw4/hw6
ATH:029C1Ch/0298CCh - BB_IQ_ADC_MEAS_3_B0 (R) ;hw4/hw6
ATH:02A8C0h - BB_IQ_ADC_MEAS_0_B1 (R) ;hw6
ATH:02A8C4h - BB_IQ_ADC_MEAS_1_B1 (R) ;hw6
ATH:02A8C8h - BB_IQ_ADC_MEAS_2_B1 (R) ;hw6
ATH:02A8CCh - BB_IQ_ADC_MEAS_3_B1 (R) ;hw6

0-31	GAIN_DC_IQ_CAL_MEAS	(R)

ATH:029C20h/02A240h - BB_RFBUS_GRANT (R) ;hw4/hw6

0	RFBUS_GRANT	(R)
1	BT_ANT	(R)

ATH:029C24h/029C20h - BB_TSTADC (R) ;hw4/hw6

0-9	TSTADC_OUT_Q	(R)
10-19	TSTADC_OUT_I	(R)

ATH:029C28h/02A368h - BB_TSTDAC (R) ;hw4/hw6

0-9	TSTDAC_OUT_Q	(R)
10-19	TSTDAC_OUT_I	(R)

ATH:029C30h/02A3B0h - BB_ILLEGAL_TX_RATE (R) ;hw4/hw6

0	ILLEGAL_TX_RATE	(R)

ATH:029C34h/0298A8h - BB_SPUR_REPORT_B0 (R) ;hw4/hw6
ATH:02A8A8h - BB_SPUR_REPORT_B1 (R) ;hw6

0-7	SPUR_EST_I	(R)
8-15	SPUR_EST_Q	(R)
16-31	POWER_WITH_SPUR_REMOVED	(R)

ATH:029C38h/02A36Ch - BB_CHANNEL_STATUS (R) ;hw4/hw6

0	BT_ACTIVE	(R)
1	RX_CLEAR_RAW	(R)
2	RX_CLEAR_MAC	(R)
3	RX_CLEAR_PAD	(R)
4-5	BB_SW_OUT_0	(R)
6-7	BB_SW_OUT_1	(R)
8-9	BB_SW_OUT_2	(R)
10-13	BB_SW_COM_OUT	(R)
14-16	ANT_DIV_CFG_USED	(R)

ATH:029C3Ch/029F80h - BB_RSSI_B0 (R) ;hw4/hw6
ATH:02AF80h - BB_RSSI_B1 (R) ;hw6
ATH:02CF80h - BB_RSSI_B3 (R) ;hw6

0-7	RSSI	(R)
8-15	RSSI_EXT	(R)

ATH:029C40h/029F84h - BB_SPUR_EST_CCK_REPORT_B0 (R) ;hw4/hw6
ATH:02AF84h - BB_SPUR_EST_CCK_REPORT_B1 (R) ;hw6

0-7	SPUR_EST_SD_I_CCK	(R)
8-15	SPUR_EST_SD_Q_CCK	(R)
16-23	SPUR_EST_I_CCK	(R)
24-31	SPUR_EST_Q_CCK	(R)

ATH:029CF0h/029CACh/02A374h - BB_CHAN_INFO_NOISE_PWR ;hw4/hw4.2/hw6

0-11	NOISE_POWER	(R)

ATH:029CF4h/029CB0h/02A378h - BB_CHAN_INFO_GAIN_DIFF ;hw4/hw4.2/hw6

0-11	FINE_PPM	(R)	
12-18	hw6: ANALOG_GAIN_DIFF_01	(R)	;-hw6

ATH:029CF8h/029CB4h/02A37Ch - BB_CHAN_INFO_FINE_TIMING ;hw4/hw4.2/hw6

```
0-11   COARSE_PPM                  (R)
12-21  FINE_TIMING                 (R)
```

ATH:029CFCh/029CB8h/02A380h - BB_CHAN_INFO_GAIN_B0 ;hw4/hw4.2/hw6
ATH:02B380h - BB_CHAN_INFO_GAIN_B1 ;hw6

```
0-7    CHAN_INFO_RSSI              (R)
8-15   CHAN_INFO_RF_GAIN           (R)
16     hw4: CHAN_INFO_XATTEN1_SW   (R)     ;\hw4 (and hw4.2)
17     hw4: CHAN_INFO_XATTEN2_SW   (R)     ;/
16-22  hw6: CHAN_INFO_MB_GAIN      (R)     ;\
23     hw6: CHAN_INFO_XATTEN1_SW   (R)     ; hw6
24     hw6: CHAN_INFO_XATTEN2_SW   (R)     ;/
```

ATH:029D00h.. - BB_CHAN_INFO_CHAN_TAB_B0[0..59] ;hw4
ATH:029CBCh.. - BB_CHAN_INFO_CHAN_TAB_B0[0..59] ;hw4.2
ATH:029B00h.. - BB_CHAN_INFO_CHAN_TAB_B0[0..59] ;hw6
ATH:02AB00h.. - BB_CHAN_INFO_CHAN_TAB_B1[0..59] ;hw6

```
0-5    hw4: MAN_Q_0    ;\          (R)     ;\
6-11   hw4: MAN_I_0    ; aka B0 ?  (R)     ;
12-15  hw4: EXP_0      ;/          (R)     ; hw4 (and hw4.2)
16-21  hw4: MAN_Q_1    ;\          (R)     ;
22-27  hw4: MAN_I_1    ; aka B1 ?  (R)     ;
28-31  hw4: EXP_1      ;/          (R)     ;/
0-31   hw6: CHANINFO_WORD          (R)     ;-hw6
```

ATH:02A000h/029E3Ch - BB_PEAK_DET_CTRL_1 ;hw4/hw6

```
0      USE_OC_GAIN_TABLE
1      USE_PEAK_DET
2-7    PEAK_DET_WIN_LEN
8-12   PEAK_DET_TALLY_THR_LOW(_0)
13-17  PEAK_DET_TALLY_THR_MED(_0)
18-22  PEAK_DET_TALLY_THR_HIGH(_0)
23-29  PEAK_DET_SETTLING
30     PWD_PKDET_DURING_CAL
31     PWD_PKDET_DURING_RX
```

ATH:02A004h/029E40h - BB_PEAK_DET_CTRL_2 ;hw4/hw6

```
0-9    RFSAT_2_ADD_RFGAIN_DEL
10-14  RF_GAIN_DROP_DB_LOW(_0)
15-19  RF_GAIN_DROP_DB_MED(_0)
20-24  RF_GAIN_DROP_DB_HIGH(_0)
25-29  RF_GAIN_DROP_DB_NON(_0)
30     hw6: ENABLE_RFSAT_RESTART
```

ATH:02A008h/029E44h - BB_RX_GAIN_BOUNDS_1 ;hw4/hw6

```
0-7    RX_MAX_MB_GAIN
8-15   RX_MAX_RF_GAIN_REF
16-23  RX_MAX_RF_GAIN
24     RX_OCGAIN_SEL_2G
25     RX_OCGAIN_SEL_5G
```

ATH:02A00Ch/029E48h - BB_RX_GAIN_BOUNDS_2 ;hw4/hw6

```
0-7    GC_RSSI_LOW_DB
8-15   RF_GAIN_REF_BASE_ADDR
16-23  RF_GAIN_BASE_ADDR
24-31  RF_GAIN_DIV_BASE_ADDR
```

ATH:02A010h/029E4Ch - BB_PEAK_DET_CAL_CTRL ;hw4/hw6

```
0-5    PKDET_CAL_WIN_THR
```

```
  6-11   PKDET_CAL_BIAS
  12-13  PKDET_CAL_MEAS_TIME_SEL
```

ATH:02A014h/029E50h - BB_AGC_DIG_DC_CTRL ;hw4/hw6

```
  0      USE_DIG_DC
  1-3    DIG_DC_SCALE_BIAS
  4-9    DIG_DC_CORRECT_CAP
  10     hw6: DIG_DC_SWITCH_CCK          ;-hw6
  16-31  DIG_DC_MIXER_SEL_MASK
```

ATH:02A018h/029F88h - BB_AGC_DIG_DC_STATUS_I_B0 (R) ;DIG_DC RES_I_0 ;hw4/hw6
ATH:02A01Ch/029F8Ch - BB_AGC_DIG_DC_STATUS_Q_B0 (R) ;DIG_DC RES_Q_0 ;hw4/hw6
ATH:02AF88h - BB_AGC_DIG_DC_STATUS_I_B1 (R) ;hw6
ATH:02AF8Ch - BB_AGC_DIG_DC_STATUS_Q_B1 (R) ;hw6

```
  0-8    DIG_DC_C1 RES_I_0 / RES_Q_0 / RES_I_1 / RES_Q_1    (R)
  9-17   DIG_DC_C2 RES_I_0 / RES_Q_0 / RES_I_1 / RES_Q_1    (R)
  18-26  DIG_DC_C3 RES_I_0 / RES_Q_0 / RES_I_1 / RES_Q_1    (R)
```

ATH:02A1F4h/02A398h - BB_BBB_TXFIR_0 ;TXFIR_COEFF_H0..H3 ;hw4/hw6
ATH:02A1F8h/02A39Ch - BB_BBB_TXFIR_1 ;TXFIR_COEFF_H4..H7 ;hw4/hw6
ATH:02A1FCh/02A3A0h - BB_BBB_TXFIR_2 ;TXFIR_COEFF_H8..H11 ;hw4/hw6

```
  0-7    TXFIR COEFF_H0 (4bit) / COEFF_H4 (6bit) / COEFF_H8 (8bit)
  8-15   TXFIR COEFF_H1 (4bit) / COEFF_H5 (6bit) / COEFF_H9 (8bit)
  16-23  TXFIR COEFF_H2 (5bit) / COEFF_H6 (7bit) / COEFF_H10 (8bit)
  24-31  TXFIR COEFF_H3 (5bit) / COEFF_H7 (7bit) / COEFF_H11 (8bit)
```

Note: The entries are aligned to 8bit boundaries, but not all entries are 8bit wide (eg. COEFF H0 and H1 are located in bit0-3 and bit8-11, with bit4-7 left unused).

ATH:02A200h/02A208h - BB_MODES_SELECT ;hw4/hw6

```
  0      CCK_MODE
  2      DYN_OFDM_CCK_MODE
  5      HALF_RATE_MODE
  6      QUARTER_RATE_MODE
  7      MAC_CLK_MODE
  8      DISABLE_DYN_CCK_DET
  9      hw6: SVD_HALF_RATE_MODE         ;\hw6
  10     hw6: DISABLE_DYN_FAST_ADC       ;/
```

ATH:02A204h/02A394h - BB_BBB_TX_CTRL ;hw4/hw6

```
  0      DISABLE_SCRAMBLER
  1      USE_SCRAMBLER_SEED
  2-3    TX_DAC_SCALE_CCK
  4      TXFIR_JAPAN_CCK
  5      ALLOW_1MBPS_SHORT
  6-8    TX_CCK_DELAY_1
  9-11   TX_CCK_DELAY_2
```

ATH:02A208h/029FC0h - BB_BBB_SIG_DETECT ;hw4/hw6

```
  0-5    WEAK_SIG_THR_CCK
  6-12   ANT_SWITCH_TIME
  13     ENABLE_ANT_FAST_DIV
  14     LB_ALPHA_128_CCK
  15     LB_RX_ENABLE_CCK
  16     CYC32_COARSE_DC_EST_CCK
  17     CYC64_COARSE_DC_EST_CCK
  18     ENABLE_COARSE_DC_CCK
  19     CYC256_FINE_DC_EST_CCK
  20     ENABLE_FINE_DC_CCK
  21     DELAY_START_SYNC_CCK
  22     USE_DC_EST_DURING_SRCH
  23     hw6: BBB_MRC_OFF_NO_SWAP                ;\hw6
```

```
  24     hw6: SWAP_DEFAULT_CHAIN_CCK            ;/
  31     ENABLE_BARKER_TWO_PHASE
```

ATH:02A20Ch/029E18h - BB_EXT_ATTEN_SWITCH_CTL_B0 ;hw4/hw6
ATH:02B20Ch/02AE18h - BB_EXT_ATTEN_SWITCH_CTL_B1 ;hw4/hw6

```
  0-5    XATTEN1_DB
  6-11   XATTEN2_DB
  12-16  XATTEN1_MARGIN
  17-21  XATTEN2_MARGIN
  22-26  hw6: XLNA_GAIN_DB                      ;-hw6
```

ATH:02A210h/029D00h - BB_BBB_RX_CTRL_1 ;hw4/hw6

```
  0-2    COARSE_TIM_THRESHOLD_2
  3-7    COARSE_TIM_THRESHOLD
  8-10   COARSE_TIM_N_SYNC
  11-15  MAX_BAL_LONG
  16-20  MAX_BAL_SHORT
  21-23  RECON_LMS_STEP
  24-30  SB_CHECK_WIN
  31     EN_RX_ABORT_CCK
```

ATH:02A214h/029D04h - BB_BBB_RX_CTRL_2 ;hw4/hw6

```
  0-5    FREQ_EST_N_AVG_LONG
  6-11   CHAN_AVG_LONG
  12-16  COARSE_TIM_THRESHOLD_3
  17-21  FREQ_TRACK_UPDATE_PERIOD
  22-25  FREQ_EST_SCALING_PERIOD
  26-31  LOOP_COEF_DPSK_C2_DATA
```

ATH:02A218h/029D08h - BB_BBB_RX_CTRL_3 ;hw4/hw6

```
  0-7    TIM_ADJUST_FREQ_DPSK
  8-15   TIM_ADJUST_FREQ_CCK
  16-23  TIMER_N_SFD
```

ATH:02A21Ch/029D0Ch - BB_BBB_RX_CTRL_4 ;hw4/hw6

```
  0-3    TIMER_N_SYNC
  4-15   TIM_ADJUST_TIMER_EXP
  16     FORCE_UNLOCKED_CLOCKS
  17     DYNAMIC_PREAM_SEL
  18     SHORT_PREAMBLE
  19-24  FREQ_EST_N_AVG_SHORT
  25-30  CHAN_AVG_SHORT
  31     hw6: USE_MRC_WEIGHT                    ;-hw6
```

ATH:02A220h/029D10h - BB_BBB_RX_CTRL_5 ;hw4/hw6

```
  0-4    LOOP_COEF_DPSK_C1_DATA
  5-9    LOOP_COEF_DPSK_C1_HEAD
  10-15  LOOP_COEF_DPSK_C2_HEAD
  16-20  LOOP_COEF_CCK_C1
  21-26  LOOP_COEF_CCK_C2
```

ATH:02A224h/029D14h - BB_BBB_RX_CTRL_6 ;hw4/hw6

```
  0-9    SYNC_START_DELAY
  10     MAP_1S_TO_2S
  11-20  START_IIR_DELAY
  21     hw6: USE_MCORR_WEIGHT                  ;\
  22     hw6: USE_BKPWR_FOR_CENTER_INDEX        ;
  23     hw6: CCK_SEL_CHAIN_BY_EO               ; hw6
  24     hw6: FORCE_CCK_SEL_CHAIN               ;
  25     hw6: FORCE_CENTER_INDEX                ;/
```

ATH:02A228h/029FC4h - BB_BBB_DAGC_CTRL ;hw4/hw6

```
0      ENABLE_DAGC_CCK
1-8    DAGC_TARGET_PWR_CCK
9      ENABLE_BARKER_RSSI_THR
10-16  BARKER_RSSI_THR
17     ENABLE_FIRSTEP_SEL
18-23  FIRSTEP_2
24-27  FIRSTEP_COUNT_LGMAX
28-29  hw6: FORCE_RX_CHAIN_CCK_0              ;\hw6
30-31  hw6: FORCE_RX_CHAIN_CCK_1              ;/
```

ATH:02A22Ch/029D18h - BB_FORCE_CLKEN_CCK ;hw4/hw6

```
0      FORCE_RX_ENABLE0
1      FORCE_RX_ENABLE1
2      FORCE_RX_ENABLE2
3      FORCE_RX_ENABLE3
4      FORCE_RX_ALWAYS
5      FORCE_TXSM_CLKEN
```

ATH:02A230h/02A250h - BB_RX_CLEAR_DELAY ;hw4/hw6

```
0-9    OFDM_XR_RX_CLEAR_DELAY
```

ATH:02A240h/029FCCh - BB_CCK_SPUR_MIT ;hw4/hw6

```
0      USE_CCK_SPUR_MIT
1-8    SPUR_RSSI_THR
9-28   CCK_SPUR_FREQ
29-30  SPUR_FILTER_TYPE
```

ATH:02A244h/02A7C0h - BB_PANIC_WATCHDOG_STATUS/BB_WATCHDOG_STATUS ;hw4/hw6

```
0-2    (PANIC_)WATCHDOG_STATUS_1
3      hw4: (PANIC_)WATCHDOG_DET_HANG         ;-hw4
3      hw6: (PANIC_)WATCHDOG_TIMEOUT          ;-hw6
4-7    (PANIC_)WATCHDOG_STATUS_2
8-11   (PANIC_)WATCHDOG_STATUS_3
12-15  (PANIC_)WATCHDOG_STATUS_4
16-19  (PANIC_)WATCHDOG_STATUS_5
20-23  (PANIC_)WATCHDOG_STATUS_6
24-27  (PANIC_)WATCHDOG_STATUS_7
28-31  (PANIC_)WATCHDOG_STATUS_8
```

ATH:02A248h/02A7C4h - BB_PANIC_WATCHDOG_CTRL_1/BB_WATCHDOG_CTRL_1 ;hw4/hw6

```
0      ENABLE_(PANIC_)WATCHDOG_(TIMEOUT_RESET_)NON_IDLE
1      ENABLE_(PANIC_)WATCHDOG_(TIMEOUT_RESET_)IDLE
2-15   (PANIC_)WATCHDOG_(TIMEOUT_RESET_)NON_IDLE_LIMIT
16-31  (PANIC_)WATCHDOG_(TIMEOUT_RESET_)IDLE_LIMIT
```

ATH:02A24Ch/02A7C8h - BB_PANIC_WATCHDOG_CTRL_2/BB_WATCHDOG_CTRL_2 ;hw4/hw6

```
0      FORCE_FAST_ADC_CLK
1      (PANIC_)WATCHDOG_(TIMEOUT_)RESET_ENA
2      (PANIC_)WATCHDOG_IRQ_ENA
```

ATH:02A250h/029FC8h - BB_IQCORR_CTRL_CCK ;hw4/hw6

```
0-4    IQCORR_Q_Q_COFF_CCK
5-10   IQCORR_Q_I_COFF_CCK
11     ENABLE_IQCORR_CCK
12-13  RXCAL_MEAS_TIME_SEL
14-15  CLCAL_MEAS_TIME_SEL
16-20  CF_CLC_INIT_RFGAIN
21     hw4.2: CF_CLC_PAL_MODE         ;-hw4.2 only (removed again in hw6)
```

ATH:02A254h/02A7CCh - BB_BLUETOOTH_CNTL ;hw4/hw6

```
0      BT_BREAK_CCK_EN
1      BT_ANT_HALT_WLAN
2      hw6:ENABLE_RFBUS_GRANT_WAKEUP          ;-hw6
```

ATH:02A258h/02A3F8h - BB_TPC_1 ;hw4/hw6

```
0      FORCE_DAC_GAIN
1-5    FORCED_DAC_GAIN
6-13   PD_DC_OFFSET_TARGET
14-15  NUM_PD_GAIN
16-17  PD_GAIN_SETTING1
18-19  PD_GAIN_SETTING2
20-21  PD_GAIN_SETTING3
22     ENABLE_PD_CALIBRATE
23-28  PD_CALIBRATE_WAIT
29     FORCE_PDADC_GAIN
30-31  FORCED_PDADC_GAIN
```

ATH:02A25Ch/02A3FCh - BB_TPC_2 ;hw4/hw6

```
0-7    TX_FRAME_TO_PDADC_ON
8-15   TX_FRAME_TO_PD_ACC_OFDM
16-23  TX_FRAME_TO_PD_ACC_CCK
```

ATH:02A260h/02A400h - BB_TPC_3 ;hw4/hw6

```
0-7    TX_END_TO_PDADC_ON
8-15   TX_END_TO_PD_ACC_ON
16-18  PD_ACC_WINDOW_DC_OFF
19-21  PD_ACC_WINDOW_CAL
22-24  PD_ACC_WINDOW_OFDM
25-27  PD_ACC_WINDOW_CCK
31     TPC_CLK_GATE_ENABLE
```

ATH:02A264h/02A404h - BB_TPC_4_B0 ;hw4/hw6
ATH:02B404h - BB_TPC_4_B1 ;hw6

```
0      PD_AVG_VALID_0/1               (R)     ;\
1-8    PD_AVG_OUT_0/1                 (R)     ;
9-13   DAC_GAIN_0/1                   (R)     ; separate settings (on hw6)
14-19  TX_GAIN_SETTING_0/1            (R)     ;
20-24  RATE_SENT_0/1                  (R)     ;/
25-30  ERROR_EST_UPDATE_POWER_THRESH          ;-global setting (not in B1)
```

ATH:02A268h/02A34Ch - BB_ANALOG_SWAP ;hw4/hw6

```
0-2    ANALOG_RX_SWAP_CNTL
3-5    ANALOG_TX_SWAP_CNTL
6      SWAP_ALT_CHN
7      ANALOG_DC_DAC_POLARITY
8      ANALOG_PKDET_DAC_POLARITY
```

ATH:02A26Ch/02A408h - BB_TPC_5_B0 ;hw4/hw6
ATH:02B408h - BB_TPC_5_B1 ;hw6

```
0-3    PD_GAIN_OVERLAP                        ;-global setting (not in B1)
4-9    PD_GAIN_BOUNDARY_1_0/1                 ;\
10-15  PD_GAIN_BOUNDARY_2_0/1                 ; separate settings (on hw6)
16-21  PD_GAIN_BOUNDARY_3_0/1                 ;
22-27  PD_GAIN_BOUNDARY_4_0/1                 ;/
```

ATH:02A270h/02A40Ch - BB_TPC_6_B0 ;hw4/hw6
ATH:02B40Ch - BB_TPC_6_B1 ;hw6

```
0-5    PD_DAC_SETTING_1
6-11   PD_DAC_SETTING_2
```

```
  12-17  PD_DAC_SETTING_3
  18-23  PD_DAC_SETTING_4
  24-25  ERROR_EST_MODE
  26-28  ERROR_EST_FILTER_COEFF
```

ATH:02A274h/02A410h - BB_TPC_7 ;hw4/hw6

```
  0-5    TX_GAIN_TABLE_MAX
  6-11   INIT_TX_GAIN_SETTING
  12     EN_CL_GAIN_MOD
  13     USE_TX_PD_IN_XPA
  14     EXTEND_TX_FRAME_FOR_TPC
  15     USE_INIT_TX_GAIN_SETTING_AFTER_WARM_RESET
```

ATH:02A280h..02A2FCh - BB_PDADC_TAB_B0[0..31] (W) ;hw4
ATH:02A480h..02A4FCh - BB_PDADC_TAB_B0[0..31] (W) ;hw6
ATH:02B480h..02B4FCh - BB_PDADC_TAB_B1[0..31] (W) ;hw6

```
  0-31   TAB_ENTRY                       (W)
```

ATH:02A300h..02A33Ch - BB_CL_TAB_B0[0..15] ;hw4
ATH:02A300h..02A33Ch - BB_CL_TAB_B0[0..15] ;hw6
ATH:02B300h..02B33Ch - BB_CL_TAB_B1[0..15] ;hw6 only (B1)

```
  0-4    CL_GAIN_MOD
  5-15   CARR_LK_DC_ADD_Q
  16-26  CARR_LK_DC_ADD_I
  27-30  BB_GAIN
```

ATH:02A340h/02A2DCh - BB_CL_MAP_0_B0 ;hw4/hw6
ATH:02A344h/02A2E0h - BB_CL_MAP_1_B0 ;hw4/hw6
ATH:02A348h/02A2E4h - BB_CL_MAP_2_B0 ;hw4/hw6
ATH:02A34Ch/02A2E8h - BB_CL_MAP_3_B0 ;hw4/hw6
ATH:02B2DCh - BB_CL_MAP_0_B1 ;hw6
ATH:02B2E0h - BB_CL_MAP_1_B1 ;hw6
ATH:02B2E4h - BB_CL_MAP_2_B1 ;hw6
ATH:02B2E8h - BB_CL_MAP_3_B1 ;hw6

```
  0-31   CL_MAP
```

ATH:02A358h/02A2D8h - BB_CL_CAL_CTRL ;hw4/hw6

```
  0      ENABLE_PARALLEL_CAL
  1      ENABLE_CL_CALIBRATE
  2-3    CF_CLC_TEST_POINT
  4-7    CF_CLC_FORCED_PAGAIN
  8-15   CARR_LEAK_MAX_OFFSET
  16-21  CF_CLC_INIT_BBGAIN
  22-29  CF_ADC_BOUND
  30     USE_DAC_CL_CORRECTION
  31     CL_MAP_HW_GEN
```

ATH:02A388h/02A244h - BB_RIFS ;hw4/hw6

```
  25     DISABLE_FCC_FIX
  26     ENABLE_RESET_TDOMAIN
  27     DISABLE_FCC_FIX2
  28     DISABLE_RIFS_CCK_FIX
  29     DISABLE_ERROR_RESET_FIX
  30     RADAR_USE_FDOMAIN_RESET
```

ATH:029934h/02A3C0h - BB_POWERTX_RATE1 ;Power TX_0..3 ;hw4/hw6
ATH:029938h/02A3C4h - BB_POWERTX_RATE2 ;Power TX_4..7 ;hw4/hw6
ATH:02A234h/02A3C8h - BB_POWERTX_RATE3 ;Power TX_1L,2L,2S ;hw4/hw6
ATH:02A238h/02A3CCh - BB_POWERTX_RATE4 ;Power TX_55L,55S,11L,11S ;hw4/hw6

```
  0-5    POWERTX_0 / POWERTX_4 / POWERTX_1L / POWERTX_55L
  8-13   POWERTX_1 / POWERTX_5 / -          / POWERTX_55S
  16-21  POWERTX_2 / POWERTX_6 / POWERTX_2L / POWERTX_11L
  24-29  POWERTX_3 / POWERTX_7 / POWERTX_2S / POWERTX_11S
```

ATH:02A38Ch/02A3D0h - BB_POWERTX_RATE5 ;Power TX HT20_0..3 ;hw4/hw6
ATH:02A390h/02A3D4h - BB_POWERTX_RATE6 ;Power TX HT20_4..7 ;hw4/hw6
ATH:02A3CCh/02A3E4h - BB_POWERTX_RATE10 ;Power TX HT20_8..11 ;hw4/hw6
ATH:02A3D0h/02A3E8h - BB_POWERTX_RATE11 ;Power TX HT20/40_12/13 ;hw4/hw6

```
  0-5    POWERTX HT20_0 / HT20_4 / HT20_8  / HT20_12
  8-13   POWERTX HT20_1 / HT20_5 / HT20_9  / HT20_13
  16-21  POWERTX HT20_2 / HT20_6 / HT20_10 / HT40_12
  24-29  POWERTX HT20_3 / HT20_7 / HT20_11 / HT40_13
```

ATH:02A3C0h/02A3D8h - BB_POWERTX_RATE7 ;Power TX HT40_0..3 ;hw4/hw6
ATH:02A3C4h/02A3DCh - BB_POWERTX_RATE8 ;Power TX HT40_4..7 ;hw4/hw6
ATH:02A3D4h/02A3ECh - BB_POWERTX_RATE12 ;Power TX HT40_8..11 ;hw4/hw6
ATH:02A3C8h/02A3E0h - BB_POWERTX_RATE9 ;PowerTX DUP40/EXT20_CCK/OFDM ;hw4/hw6

```
  0-5    POWERTX HT40_0 / HT40_4 / HT40_8  / DUP40_CCK
  8-13   POWERTX HT40_1 / HT40_5 / HT40_9  / DUP40_OFDM
  16-21  POWERTX HT40_2 / HT40_6 / HT40_10 / EXT20_CCK
  24-29  POWERTX HT40_3 / HT40_7 / HT40_11 / EXT20_OFDM
```

ATH:02A3BCh/02A3F4h - BB_POWERTX_SUB ;Power TX Sub_for_2chain ;hw4/hw6

```
  0-5    POWERTX SUB_FOR_2CHAIN
```

ATH:02A278h/02A414h - BB_TPC_8 ;DESIRED_SCALE_0..5 ;hw4/hw6

```
  0-4    DESIRED_SCALE_0
  5-9    DESIRED_SCALE_1
  10-14  DESIRED_SCALE_2
  15-19  DESIRED_SCALE_3
  20-24  DESIRED_SCALE_4
  25-29  DESIRED_SCALE_5
```

ATH:02A27Ch/02A418h - BB_TPC_9 ;DESIRED_SCALE_6,7,CCK and MISC ;hw4/hw6

```
  0-4    DESIRED_SCALE_6
  5-9    DESIRED_SCALE_7
  10-14  DESIRED_SCALE_CCK
  20     EN_PD_DC_OFFSET_THR
  21-26  PD_DC_OFFSET_THR
  27-30  WAIT_CALTX_SETTLE
  31     DISABLE_PDADC_RESIDUAL_DC_REMOVAL
```

ATH:02A394h/02A41Ch - BB_TPC_10 ;DESIRED_SCALE HT20_0..5 ;hw4/hw6
ATH:02A3E4h/02A42Ch - BB_TPC_14 ;DESIRED_SCALE HT20_8..13 ;hw4/hw6
ATH:02A3DCh/02A424h - BB_TPC_12 ;DESIRED_SCALE HT40_0..5 ;hw4/hw6
ATH:02A3E0h/02A428h - BB_TPC_13 ;DESIRED_SCALE HT40_6..7 ;hw4/hw6
ATH:02A3E8h/02A430h - BB_TPC_15 ;DESIRED_SCALE HT40_8..13 ;hw4/hw6

```
  0-4    DESIRED_SCALE HT20_0 / HT20_8  / HT40_0 / HT40_6 / HT40_8
  5-9    DESIRED_SCALE HT20_1 / HT20_9  / HT40_1 / HT40_7 / HT40_9
  10-14  DESIRED_SCALE HT20_2 / HT20_10 / HT40_2 / -      / HT40_10
  15-19  DESIRED_SCALE HT20_3 / HT20_11 / HT40_3 / -      / HT40_11
  20-24  DESIRED_SCALE HT20_4 / HT20_12 / HT40_4 / -      / HT40_12
  25-29  DESIRED_SCALE HT20_5 / HT20_13 / HT40_5 / -      / HT40_13
```

ATH:02A398h/02A420h - BB_TPC_11_B0 ;DESIRED_SCALE HT20_6..7 and OLPC ;hw4/hw6
ATH:02B420h - BB_TPC_11_B1 ;hw6

```
  0-4    DESIRED_SCALE HT20_6                ;\global setting (not in B1)
  5-9    DESIRED_SCALE HT20_7                ;/
  16-23  OLPC_GAIN_DELTA_0/1                 ;\
```

```
  24-31  hw4: OLPC_GAIN_DELTA_0/1_PAL_ON  ;-hw4 ; separate settings (on hw6)
  24-25  hw6: OLPC_GAIN_DELTA_0/1_LSB_EXT ;-hw6 ;/
```

ATH:02A39Ch/02A2C0h - BB_CAL_CHAIN_MASK ;hw4/hw6
```
  0-2    CAL_CHAIN_MASK
```

ATH:02A3D8h/02A358h - BB_FORCE_ANALOG ;hw4/hw6
```
  0      FORCE_XPAON
  1-3    FORCED_XPAON
  4      FORCE_PDADC_PWD
  5-7    FORCED_PDADC_PWD
```

ATH:02A3ECh/02A434h - BB_TPC_16 ;hw4/hw6
```
  8-13   PDADC_PAR_CORR_CCK
  16-21  PDADC_PAR_CORR_OFDM
  24-29  PDADC_PAR_CORR_HT40
```

ATH:02A3F0h/02A438h - BB_TPC_17 ;hw4/hw6
```
  0      ENABLE_PAL
  1      ENABLE_PAL_CCK
  2      ENABLE_PAL_OFDM_20
  3      ENABLE_PAL_OFDM_40
  4-9    PAL_POWER_THRESHOLD
  10     FORCE_PAL_LOCKED
  11-16  INIT_TX_GAIN_SETTING_PAL_ON
```

ATH:02A3F4h/02A43Ch - BB_TPC_18 ;hw4/hw6
```
  0-7    THERM_CAL_VALUE
  8-15   VOLT_CAL_VALUE
  16     USE_LEGACY_TPC
  17-22  hw6: MIN_POWER_THERM_VOLT_GAIN_CORR    ;-hw6
```

ATH:02A3F8h/02A440h - BB_TPC_19/BB_TPC_19_B0 ;hw4/hw6
ATH:02B440h - BB_TPC_19_B1 ;hw6
```
  0-7    ALPHA_THERM
  8-15   ALPHA_THERM_PAL_ON
  16-20  ALPHA_VOLT
  21-25  ALPHA_VOLT_PAL_ON
```

ATH:02A3FCh/02A444h - BB_TPC_20 ;hw4/hw6
```
  0-23   ENABLE_PAL_MCS_0..23
```

ATH:02A518h..02A554h - BB_CALTX_GAIN_SET_(0,2,4,6,..,28,30) ;hw4
ATH:02A600h..02A63Ch - BB_CALTX_GAIN_SET_(0,2,4,6,..,28,30) ;hw6
Contains 32 table entries (numbered 0..31), with two 14bit entries per word.
```
  0-13   CALTX_GAIN_SET_nn   table entry 0,2,4,6,...,28,30 accordingly
  14-27  CALTX_GAIN_SET_nn   table entry 1,3,5,7,...,29,31 accordingly
  28-31  -
```

ATH:02A400h..02A47Ch - BB_TX_GAIN_TAB_(1..32) ;hw4
ATH:02A500h..02A57Ch - BB_TX_GAIN_TAB_(1..32) ;hw6
Contains 32 table entries (numbered 1..32), with one 32bit entry per word.
```
  0-31   TG_TABLE entry   entry 1..32 accordingly
```
On hw6, the 32bit entries are expanded to 34bit size (with extra 2bits in the BB_TX_GAIN_TAB_xxx_LSB_EXT registers).

ATH:02A58Ch - BB_TX_GAIN_TAB_1_16_LSB_EXT ;hw6
ATH:02A590h - BB_TX_GAIN_TAB_17_32_LSB_EXT ;hw6
Contains 32 table entries (numbered 1..32), with sixteen 2bit entries per word.

```
  0-31   TG_TABLE_LSB_EXT  (sixteen 2bit entries per word)
```
These 2bit values are used to expand the 32bit entries in BB_TX_GAIN_TAB_(1..32) to 34bit size.

ATH:02A6DCh/02A644h - BB_TXIQCAL_CONTROL_0 ;hw4/hw6
```
  0      IQC_TX_TABLE_SEL
  1-6    BASE_TX_TONE_DB
  7-12   MAX_TX_TONE_GAIN
  13-18  MIN_TX_TONE_GAIN
  19-22  CALTXSHIFT_DELAY
  23-29  LOOPBACK_DELAY
  30     hw6: ENABLE_COMBINED_CARR_IQ_CAL       ;\hw6
  31     hw6: ENABLE_TXIQ_CALIBRATE             ;/
```

ATH:02A6E0h/02A648h - BB_TXIQCAL_CONTROL_1 ;hw4/hw6
```
  0-5    RX_INIT_GAIN_DB
  6-11   MAX_RX_GAIN_DB
  12-17  MIN_RX_GAIN_DB
  18-24  IQCORR_I_Q_COFF_DELPT
```

ATH:02A6E4h/02A64Ch - BB_TXIQCAL_CONTROL_2 ;hw4/hw6
```
  0-3    IQC_FORCED_PAGAIN
  4-8    IQCAL_MIN_TX_GAIN
  9-13   IQCAL_MAX_TX_GAIN
```

ATH:02A6E8h/0298B0h - BB_TXIQCAL_CONTROL_3 ;hw4/hw6
```
  0-5    PWR_HIGH_DB
  6-11   PWR_LOW_DB
  12-21  IQCAL_TONE_PHS_STEP
  22-23  DC_EST_LEN
  24     ADC_SAT_LEN
  25-26  ADC_SAT_SEL
  27-28  IQCAL_MEAS_LEN
  29-30  DESIRED_SIZE_DB
  31     TX_IQCORR_EN
```

ATH:02A6ECh/02A650h - BB_TXIQ_CORR_COEFF_01_B0 ;hw4/hw6
ATH:02A6F0h/02A654h - BB_TXIQ_CORR_COEFF_23_B0 ;hw4/hw6
ATH:02A6F4h/02A658h - BB_TXIQ_CORR_COEFF_45_B0 ;hw4/hw6
ATH:02A6F8h/02A65Ch - BB_TXIQ_CORR_COEFF_67_B0 ;hw4/hw6
ATH:02A6FCh/02A660h - BB_TXIQ_CORR_COEFF_89_B0 ;hw4/hw6
ATH:02A700h/02A664h - BB_TXIQ_CORR_COEFF_AB_B0 ;hw4/hw6
ATH:02A704h/02A668h - BB_TXIQ_CORR_COEFF_CD_B0 ;hw4/hw6
ATH:02A708h/02A66Ch - BB_TXIQ_CORR_COEFF_EF_B0 ;hw4/hw6
ATH:02B650h - BB_TXIQ_CORR_COEFF_01_B1 ;hw6
ATH:02B654h - BB_TXIQ_CORR_COEFF_23_B1 ;hw6
ATH:02B658h - BB_TXIQ_CORR_COEFF_45_B1 ;hw6
ATH:02B65Ch - BB_TXIQ_CORR_COEFF_67_B1 ;hw6
ATH:02B660h - BB_TXIQ_CORR_COEFF_89_B1 ;hw6
ATH:02B664h - BB_TXIQ_CORR_COEFF_AB_B1 ;hw6
ATH:02B668h - BB_TXIQ_CORR_COEFF_CD_B1 ;hw6
ATH:02B66Ch - BB_TXIQ_CORR_COEFF_EF_B1 ;hw6
The B0 (and B1) table each contain 16 entries (numbered 0..F), with two 14bit entries per word.
```
  0-13   IQC_COEFF_TABLE_n      ;table entry (n=0,2,4,6,8,A,C,E) accordingly
  14-27  IQC_COEFF_TABLE_n      ;table entry (n=1,3,5,7,9,B,D,F) accordingly
  28-31  -
```

ATH:02A70Ch/02A670h - BB_CAL_RXBB_GAIN_TBL_0 ;hw4/hw6
ATH:02A710h/02A674h - BB_CAL_RXBB_GAIN_TBL_4 ;hw4/hw6
ATH:02A714h/02A678h - BB_CAL_RXBB_GAIN_TBL_8 ;hw4/hw6

ATH:02A718h/02A67Ch - BB_CAL_RXBB_GAIN_TBL_12 ;hw4/hw6
ATH:02A71Ch/02A680h - BB_CAL_RXBB_GAIN_TBL_16 ;hw4/hw6
ATH:02A720h/02A684h - BB_CAL_RXBB_GAIN_TBL_20 ;hw4/hw6
ATH:02A724h/02A688h - BB_CAL_RXBB_GAIN_TBL_24 ;hw4/hw6
Contains 25 table entries (numbered 0..24), with four 6bit entries per word (except, only one entry in the last word).

```
  0-5    TXCAL_RX_BB_GAIN_TABLE_n   ;table entry (n=0,4, 8,12,16,20,24)
  6-11   TXCAL_RX_BB_GAIN_TABLE_n   ;table entry (n=1,5, 9,13,17,21)
  12-17  TXCAL_RX_BB_GAIN_TABLE_n   ;table entry (n=2,6,10,14,18,22)
  18-23  TXCAL_RX_BB_GAIN_TABLE_n   ;table entry (n=3,7,11,15,19,23)
  24-31  -
```

ATH:02A728h/02A68Ch - BB_TXIQCAL_STATUS_B0 (R) ;hw4/hw6
ATH:02B68Ch - BB_TXIQCAL_STATUS_B1 (R) ;hw6

```
  0      TXIQCAL_FAILED               (R)
  1-5    CALIBRATED_GAINS             (R)
  6-11   TONE_GAIN_USED               (R)
  12-17  RX_GAIN_USED                 (R)
  18-24  hw4: LAST_MEAS_ADDR (7bit)   (R)     ;-hw4
  18-23  hw6: LAST_MEAS_ADDR (6bit)   (R)     ;-hw6
```

ATH:02A7D8h/02A2CCh - BB_FCAL_1 ;hw4/hw6

```
  0-9    FLC_PB_FSTEP
  10-19  FLC_SB_FSTEP
  20-24  FLC_PB_ATTEN
  25-29  FLC_SB_ATTEN
```

ATH:02A7DCh/02A2D0h - BB_FCAL_2_B0 ;hw4/hw6
ATH:02B2D0h - BB_FCAL_2_B1 ;hw6

```
  0-2    FLC_PWR_THRESH                   ;-global setting (not in B1)
  3-7    FLC_SW_CAP_VAL_0/1               ;-separate settings (on hw6)
  8-9    FLC_BBMISCGAIN                   ;\
  10-12  FLC_BB1DBGAIN                    ;
  13-14  FLC_BB6DBGAIN                    ; global setting (not in B1)
  15     FLC_SW_CAP_SET                   ;
  16-18  FLC_MEAS_WIN                     ;/
  20-24  FLC_CAP_VAL_STATUS_0/1    (R)    ;-separate settings (on hw6)
```

ATH:02A7E0h/02A22Ch - BB_RADAR_BW_FILTER ;hw4/hw6

```
  0      RADAR_AVG_BW_CHECK
  1      RADAR_DC_SRC_SEL
  2-3    RADAR_FIRPWR_SEL
  4-5    RADAR_PULSE_WIDTH_SEL
  8-14   RADAR_DC_FIRPWR_THRESH
  15-20  RADAR_DC_PWR_BIAS
  21-26  RADAR_BIN_MAX_BW
```

ATH:02A7E4h/02A2D4h - BB_DFT_TONE_CTRL_B0 ;hw4/hw6
ATH:02B2D4h - BB_DFT_TONE_CTRL_B1 ;hw6

```
  0      DFT_TONE_EN
  2-3    DFT_TONE_AMP_SEL
  4-12   DFT_TONE_FREQ_ANG
```

ATH:02A7E8h/02A448h - BB_THERM_ADC_1 ;hw4/hw6

```
  0-7    INIT_THERM_SETTING
  8-15   INIT_VOLT_SETTING
  16-23  INIT_ATB_SETTING
  24-25  SAMPLES_CNT_CODING
  26     USE_INIT_THERM_VOLT_ATB_AFTER_WARM_RESET
  27     FORCE_THERM_VOLT_ATB_TO_INIT_SETTINGS
```

```
  28      hw6: CHECK_DONE_FOR_1ST_ADC_MEAS_OF_EACH_FRAME    ;\hw6
  29      hw6: THERM_MEASURE_RESET                          ;/
```

ATH:02A7ECh/02A44Ch - BB_THERM_ADC_2 ;hw4/hw6

```
  0-11    MEASURE_THERM_FREQ
  12-21   MEASURE_VOLT_FREQ
  22-31   MEASURE_ATB_FREQ
```

ATH:02A7F0h/02A450h - BB_THERM_ADC_3 ;hw4/hw6

```
  0-7     THERM_ADC_OFFSET
  8-16    THERM_ADC_SCALED_GAIN
  17-29   ADC_INTERVAL
```

ATH:02A7F4h/02A454h - BB_THERM_ADC_4 ;hw4/hw6

```
  0-7     LATEST_THERM_VALUE              (R)
  8-15    LATEST_VOLT_VALUE               (R)
  16-23   LATEST_ATB_VALUE                (R)
  24      hw6: FORCE_THERM_CHAIN                            ;\hw6
  25-27   hw6: PREFERRED_THERM_CHAIN                        ;/
```

ATH:02A7F8h/02A458h - BB_TX_FORCED_GAIN ;hw4/hw6

```
  0       FORCE_TX_GAIN
  1-3     FORCED_TXBB1DBGAIN
  4-5     FORCED_TXBB6DBGAIN
  6-9     FORCED_TXMXRGAIN
  10-13   FORCED_PADRVGNA
  14-17   FORCED_PADRVGNB
  18-21   FORCED_PADRVGNC
  22-23   FORCED_PADRVGND
  24      FORCED_ENABLE_PAL
  25-27   hw6: FORCED_OB                            ;\
  28-30   hw6: FORCED_DB                            ; hw6
  31      hw6: FORCED_GREEN_PAPRD_ENABLE            ;/
```

ATH:02A7FCh/02A7DCh - BB_ECO_CTRL ;hw4/hw6

```
  0-7     ECO_CTRL
```

```
 _______________ below in hw4.2 and hw6 only (not original hw4) _______________
```

ATH:029DE4h/0298E4h - BB_PAPRD_AM2AM_MASK ;hw4.2/hw6 (not original hw4)

```
  0-24    PAPRD_AM2AM_MASK                          ;-newer revision only
```

ATH:029DE8h/0298E8h - BB_PAPRD_AM2PM_MASK ;hw4.2/hw6 (not original hw4)

```
  0-24    PAPRD_AM2PM_MASK                          ;-newer revision only
```

ATH:029DECh/0298ECh - BB_PAPRD_HT40_MASK ;hw4.2/hw6 (not original hw4)

```
  0-24    PAPRD_HT40_MASK                           ;-newer revision only
```

ATH:029DF0h - BB_PAPRD_CTRL0 ;hw4.2 (not original hw4)
ATH:0298F0h - BB_PAPRD_CTRL0_B0 ;hw6
ATH:02A8F0h - BB_PAPRD_CTRL0_B1 ;hw6

```
  0       PAPRD_ENABLE                              ;\
  1       PAPRD_ADAPTIVE_USE_SINGLE_TABLE           ; newer revision only
  2-26    PAPRD_VALID_GAIN                          ;
  27-31   PAPRD_MAG_THRSH                           ;/
```

ATH:029DF4h - BB_PAPRD_CTRL1 ;hw4.2 (not original hw4)
ATH:0298F4h - BB_PAPRD_CTRL1_B0 ;hw6
ATH:02A8F4h - BB_PAPRD_CTRL1_B1 ;hw6

```
  0       PAPRD_ADAPTIVE_SCALING_ENABLE             ;\
```

```
  1      PAPRD_ADAPTIVE_AM2AM_ENABLE           ;
  2      PAPRD_ADAPTIVE_AM2PM_ENABLE           ; newer revision only
  3-8    PAPRD_POWER_AT_AM2AM_CAL              ;
  9-16   PA_GAIN_SCALE_FACTOR                  ;
  17-26  PAPRD_MAG_SCALE_FACTOR                ;
  27     PAPRD_TRAINER_IANDQ_SEL               ;/
```

ATH:029DF8h - BB_PA_GAIN123 ;hw4.2 (not original hw4)
ATH:0298F8h - BB_PA_GAIN123_B0 ;hw6
ATH:02A8F8h - BB_PA_GAIN123_B1 ;hw6

```
  0-9    PA_GAIN1                              ;\
  10-19  PA_GAIN2                              ; newer revision only
  20-29  PA_GAIN3                              ;/
```

ATH:029DFCh - BB_PA_GAIN45 ;hw4.2 (not original hw4)
ATH:0298FCh - BB_PA_GAIN45_B0 ;hw6
ATH:02A8FCh - BB_PA_GAIN45_B1 ;hw6

```
  0-9    PA_GAIN4                              ;\
  10-19  PA_GAIN5                              ; newer revision only
  20-24  PAPRD_ADAPTIVE_TABLE_VALID            ;/
```

ATH:029E00h..029E1Ch - BB_PAPRD_PRE_POST_SCALE_(0..7) ;hw4.2 (not hw4)
ATH:029900h..02991Ch - BB_PAPRD_PRE_POST_SCALE_(0..7)_B0 ;hw6
ATH:02A900h..02A91Ch - BB_PAPRD_PRE_POST_SCALE_(0..7)_B1 ;hw6

```
  0-17   PAPRD_PRE_POST_SCALING                ;-newer revision only
```

ATH:029E20h.. - BB_PAPRD_MEM_TAB[.....] ;hw4.2 (not original hw4)
ATH:029920h.. - BB_PAPRD_MEM_TAB_B0[0..119] ;hw6
ATH:02A920h.. - BB_PAPRD_MEM_TAB_B1[0..119] ;hw6

```
  0-21   PAPRD_MEM                             ;-newer revision only
```

ATH:02A35Ch/02A2ECh - BB_CL_MAP_PAL_0_B0 ;hw4.2/hw6 (not original hw4)
ATH:02A360h/02A2F0h - BB_CL_MAP_PAL_1_B0 ;hw4.2/hw6 (not original hw4)
ATH:02A364h/02A2F4h - BB_CL_MAP_PAL_2_B0 ;hw4.2/hw6 (not original hw4)
ATH:02A368h/02A2F8h - BB_CL_MAP_PAL_3_B0 ;hw4.2/hw6 (not original hw4)
ATH:02B2ECh - BB_CL_MAP_PAL_0_B1 ;hw6
ATH:02B2F0h - BB_CL_MAP_PAL_1_B1 ;hw6
ATH:02B2F4h - BB_CL_MAP_PAL_2_B1 ;hw6
ATH:02B2F8h - BB_CL_MAP_PAL_3_B1 ;hw6

```
  0-31   CL_MAP_PAL
```

ATH:02A72Ch/02A690h - BB_PAPRD_TRAINER_CNTL1 ;hw4.2/hw6 (not original hw4)

```
  0      CF_PAPRD_TRAIN_ENABLE                 ;\
  1-7    CF_PAPRD_AGC2_SETTLING                ;
  8      CF_PAPRD_IQCORR_ENABLE                ; newer revision only
  9      CF_PAPRD_RX_BB_GAIN_FORCE             ;
  10     CF_PAPRD_TX_GAIN_FORCE                ;
  11     CF_PAPRD_LB_ENABLE                    ;
  12-18  CF_PAPRD_LB_SKIP                      ;/
```

ATH:02A730h/02A694h - BB_PAPRD_TRAINER_CNTL2 ;hw4.2/hw6 (not original hw4)

```
  0-31   CF_PAPRD_INIT_RX_BB_GAIN              ;-newer revision only
```

ATH:02A734h/02A698h - BB_PAPRD_TRAINER_CNTL3 ;hw4.2/hw6 (not original hw4)

```
  0-5    CF_PAPRD_ADC_DESIRED_SIZE             ;\
  6-11   CF_PAPRD_QUICK_DROP                   ;
  12-16  CF_PAPRD_MIN_LOOPBACK_DEL             ;
  17-19  CF_PAPRD_NUM_CORR_STAGES              ; newer revision only
  20-23  CF_PAPRD_COARSE_CORR_LEN              ;
```

```
  24-27  CF_PAPRD_FINE_CORR_LEN                ;/
  28     hw4.2: CF_PAPRD_BBTXMIX_DISABLE       ;-hw4.2
  28     hw6: CF_PAPRD_REUSE_CORR              ;\hw6
  29     hw6: CF_PAPRD_BBTXMIX_DISABLE         ;/
```

ATH:02A738h/02A69Ch - BB_PAPRD_TRAINER_CNTL4 ;hw4.2/hw6 (not original hw4)

```
  0-11   CF_PAPRD_MIN_CORR                     ;\
  12-15  CF_PAPRD_SAFETY_DELTA                 ; newer revision only
  16-25  CF_PAPRD_NUM_TRAIN_SAMPLES            ;/
```

ATH:02A73Ch/02A6A0h - BB_PAPRD_TRAINER_STAT1 ;hw4.2/hw6 (not original hw4)

```
  0      PAPRD_TRAIN_DONE                      ;\
  1      PAPRD_TRAIN_INCOMPLETE        (R)     ;
  2      PAPRD_CORR_ERR                (R)     ; newer revision only
  3      PAPRD_TRAIN_ACTIVE            (R)     ;
  4-8    PAPRD_RX_GAIN_IDX             (R)     ;
  9-16   PAPRD_AGC2_PWR                (R)     ;/
```

ATH:02A740h/02A6A4h - BB_PAPRD_TRAINER_STAT2 ;hw4.2/hw6 (not original hw4)

```
  0-15   PAPRD_FINE_VAL                (R)     ;\
  16-20  PAPRD_COARSE_IDX              (R)     ; newer revision only
  21-22  PAPRD_FINE_IDX                (R)     ;/
```

ATH:02A744h/02A6A8h - BB_PAPRD_TRAINER_STAT3 ;hw4.2/hw6 (not original hw4)

```
  0-19   PAPRD_TRAIN_SAMPLES_CNT       (R)     ;-newer revision only
```

_____ below on hw4 only _____

ATH:02A480h..02A4FCh - BB_TX_GAIN_TAB_PAL_(1..32) ;hw4 only (not hw6)

Contains 32 table entries (numbered 1..32), with one 32bit entry per word.

```
  0-31   TG_TABLE_PAL_ON   entry 1..32 accordingly
```

Seems to be some extra table, alternately to "BB_TX_GAIN_TAB_(1..32)". In hw6, this has been replaced by the "LSB_EXT" feature (see BB_TPC_11).

ATH:02A558h..02A6D4h - BB_TXIQCAL_MEAS_B0[0..95] (R) ;hw4 only (not hw6)

```
  0-11   TXIQC_MEAS_DATA0_0            (R)   ;entry 0,2,4,...,190 (?)
  12-23  TXIQC_MEAS_DATA1_0            (R)   ;entry 1,3,5,...,191 (?)
  24-31  -
```

ATH:02A6D8h - BB_TXIQCAL_START ;hw4 only (not hw6)

```
  0      DO_TX_IQCAL
```

_____ below on hw6 only _____

ATH:029C04h - BB_LDPC_CNTL1 ;hw6

```
  0-31   LDPC_LLR_SCALING0
```

ATH:029C08h - BB_LDPC_CNTL2 ;hw6

```
  0-15   LDPC_LLR_SCALING1
  16-26  LDPC_LATENCY
```

ATH:029C18h - BB_ML_CNTL1 ;hw6

```
  0-23   CF_ML_2S_WEIGHT_TABLE
  24-25  CF_IS_FLAT_CH_THR_ML
  26-27  CF_IS_FLAT_CH_THR_ZF
```

ATH:029C1Ch - BB_ML_CNTL2 ;hw6

```
  0-23   CF_ML_3S_WEIGHT_TABLE
```

ATH:029E54h - BB_BT_COEX_1 ;hw6

0-4	PEAK_DET_TALLY_THR_LOW_1
5-9	PEAK_DET_TALLY_THR_MED_1
10-14	PEAK_DET_TALLY_THR_HIGH_1
15-19	RF_GAIN_DROP_DB_LOW_1
20-24	RF_GAIN_DROP_DB_MED_1
25-29	RF_GAIN_DROP_DB_HIGH_1
30	BT_TX_DISABLE_NF_CAL

ATH:029E58h - BB_BT_COEX_2 ;hw6

0-4	PEAK_DET_TALLY_THR_LOW_2
5-9	PEAK_DET_TALLY_THR_MED_2
10-14	PEAK_DET_TALLY_THR_HIGH_2
15-19	RF_GAIN_DROP_DB_LOW_2
20-24	RF_GAIN_DROP_DB_MED_2
25-29	RF_GAIN_DROP_DB_HIGH_2
30-31	RFSAT_RX_RX

ATH:029E5Ch - BB_BT_COEX_3 ;hw6

0-1	RFSAT_BT_SRCH_SRCH
2-3	RFSAT_BT_RX_SRCH
4-5	RFSAT_BT_SRCH_RX
6-7	RFSAT_WLAN_SRCH_SRCH
8-9	RFSAT_WLAN_RX_SRCH
10-11	RFSAT_WLAN_SRCH_RX
12-13	RFSAT_EQ_SRCH_SRCH
14-15	RFSAT_EQ_RX_SRCH
16-17	RFSAT_EQ_SRCH_RX
18-22	RF_GAIN_DROP_DB_NON_1
23-27	RF_GAIN_DROP_DB_NON_2
28-31	BT_RX_FIRPWR_INCR

ATH:029E60h - BB_BT_COEX_4 ;RFGAIN_EQV_LNA_0..3 ;hw6
ATH:029E64h - BB_BT_COEX_5 ;RFGAIN_EQV_LNA_4..7 ;hw6

0-7	RFGAIN_EQV_LNA_0 / RFGAIN_EQV_LNA_4
8-15	RFGAIN_EQV_LNA_1 / RFGAIN_EQV_LNA_5
16-23	RFGAIN_EQV_LNA_2 / RFGAIN_EQV_LNA_6
24-31	RFGAIN_EQV_LNA_3 / RFGAIN_EQV_LNA_7

ATH:029E68h - BB_REDPWR_CTRL_1 ;hw6

0-1	REDPWR_MODE
2	REDPWR_MODE_CLR
3	REDPWR_MODE_SET
4-8	GAIN_CORR_DB2
9-12	SCFIR_ADJ_GAIN
13-17	QUICKDROP_RF
18	BYPASS_FIR_F
19	ADC_HALF_REF_F

ATH:029E6Ch - BB_REDPWR_CTRL_2 ;hw6

0-6	SC01_SW_INDEX
7-13	SC10_SW_INDEX
14-20	LAST_SC0_INDEX

ATH:029FD0h - BB_MRC_CCK_CTRL ;hw6

0	BBB_MRC_EN
1	AGCDP_CCK_MRC_MUX_REG
2-4	AGCDP_CCK_PD_ACCU_THR_HI
5-7	AGCDP_CCK_PD_ACCU_THR_LOW
8-11	AGCDP_CCK_BARKER_RSSI_THR
12-16	AGCDP_CCK_MRC_BK_THR_HI

```
  17-21  AGCDP_CCK_MRC_BK_THR_LOW
  22-27  AGCDP_CCK_MIN_VALUE
```

ATH:029FD4h - BB_CCK_BLOCKER_DET ;hw6

```
  0      CCK_FREQ_SHIFT_BLOCKER_DETECTION
  1      CCK_BLOCKER_DET_RESTART_WEAK_SIG
  2-5    CCK_BLOCKER_DET_BKSUM_NUM
  6-8    BK_VALID_DELAY
  9-13   CCK_BLOCKER_DET_THR
  14-19  CCK_BLOCKER_DET_DELAY_THR
  20-25  CCK_BLOCKER_MONITOR_TIME
  26     SKIP_RAMP_ENABLE
  27-31  CCK_DET_RAMP_THR
```

ATH:02A384h - BB_SM_HIST ;hw6

```
  0      SM_REC_EN
  1      SM_REC_MODE
  2-3    SM_REC_TIME_RES
  4-11   SM_REC_PART_EN
  12-14  SM_REC_CHN_EN
  15-18  SM_REC_DATA_NUM
  19     SM_REC_AGC_SEL
  20-22  SM_REC_MAC_TRIG
  24-29  SM_REC_LAST_ADDR               (R)
```

ATH:02A580h - BB_RTT_CTRL ;hw6

```
  0      ENA_RADIO_RETENTION
  1-6    RESTORE_MASK
  7      FORCE_RADIO_RESTORE
```

ATH:0298BCh - BB_GREEN_TX_CONTROL_1 ;hw6

```
  0      GREEN_TX_ENABLE
  1      GREEN_CASES
```

ATH:02D800h - BB_MIT_RF_CNTL ;hw6

```
  0      MIT_FORCE_SYNTH_ON
  1      MIT_FORCE_SYNTH_ON_EN
  2      MIT_FORCE_ACTIVE_ON
```

ATH:02D804h - BB_MIT_CCA_CNTL ;hw6

```
  0-2    MIT_CCA_MODE_SEL
  3-20   MIT_CCA_COUNT
```

ATH:02D808h - BB_MIT_RSSI_CNTL_1 ;hw6

```
  0-5    MIT_RSSI_TH
  6-11   MIT_RX_RF_ATT_TH_H
  12-17  MIT_RX_RF_ATT_TH_L
  18-23  MIT_RX_RF_ATT_OFFSET
  24-29  MIT_AGC_LIMIT
```

ATH:02D80Ch - BB_MIT_RSSI_CNTL_2 ;hw6

```
  0      MIT_AGC_SEL
  1-11   MIT_RSSI_BASE
```

ATH:02D810h - BB_MIT_TX_CNTL ;hw6

```
  0-7    MIT_TX_STA_CNT
  8-21   MIT_TX_END_DLY_CNT
  22     MIT_TX_THROUGH_ENA
  23-25  MIT_TXHDR_CHAIN_MASK_CCK
  26-28  MIT_TXHDR_PAPRD_TRAIN_MASK_CCK
  29-30  MIT_TXHDR_CHAN_MODE_CCK
```

ATH:02D814h - BB_MIT_RX_CNTL ;hw6

0-7	MIT_RX_END_DLY_CNT	
8	MIT_RX_THROUGH_ENA	

ATH:02D818h - BB_MIT_OUT_CNTL ;hw6

0-1	MIT_CLK_TUNE_MOD	
2	MIT_NO_DATA_TO_ATH	

ATH:02D81Ch - BB_MIT_SPARE_CNTL ;hw6

0-30	MIT_SPARE_IN	
31	MIT_SPARE_OUT	(R)

ATH:029F90h - BB_DC_CAL_STATUS_B0 (R) ;hw6
ATH:02AF90h - BB_DC_CAL_STATUS_B1 (R) ;hw6

0-4	OFFSETC1I	(R)
5-9	OFFSETC1Q	(R)
10-14	OFFSETC2I	(R)
15-19	OFFSETC2Q	(R)
20-24	OFFSETC3I	(R)
25-29	OFFSETC3Q	(R)

ATH:02A584h - BB_RTT_TABLE_SW_INTF_B0 ;hw6
ATH:02B584h - BB_RTT_TABLE_SW_INTF_B1 ;hw6

0	SW_RTT_TABLE_ACCESS	
1	SW_RTT_TABLE_WRITE	
2-4	SW_RTT_TABLE_ADDR	

ATH:02A588h - BB_RTT_TABLE_SW_INTF_1_B0 ;hw6
ATH:02B588h - BB_RTT_TABLE_SW_INTF_1_B1 ;hw6

4-31	SW_RTT_TABLE_DATA	

ATH:02A7F0h - BB_TABLES_INTF_ADDR_B0 ;hw6
ATH:02B7F0h - BB_TABLES_INTF_ADDR_B1 ;hw6
ATH:029BF0h - BB_CHN_TABLES_INTF_ADDR ;hw6
ATH:02ABF0h - BB_CHN1_TABLES_INTF_ADDR ;hw6

2-17	TABLES_ADDR	
31	ADDR_AUTO_INCR	

ATH:02A7F4h - BB_TABLES_INTF_DATA_B0 ;hw6
ATH:02B7F4h - BB_TABLES_INTF_DATA_B1 ;hw6
ATH:029BF4h - BB_CHN_TABLES_INTF_DATA ;hw6
ATH:02ABF4h - BB_CHN1_TABLES_INTF_DATA ;hw6

0-31	TABLES_DATA	

ATH:02CE00h - BB_DUMMY (R) ;hw6
ATH:02C800h - BB_DUMMY1[0..255] (R) ;hw6
ATH:02D200h - BB_DUMMY2[0..383] (R) ;hw6

0	DUMMY	(R)

Whatever "dummy" registers, maybe intended only for padding purposes; to get the three BB_RSSI_B0/B1/B3 registers mapped to 029F80h+(0,1,3)*1000h on hw6.

_____ missing B1 registers in hw4 _____

hw4 does have only a few "B1" registers defined, although it does have a lot of "B0" registers - which would suggest the presence of corresponding "B1" registers, but those seem to exist only on hw6.
Maybe the hw4 designed was already aimed at adding "B1" registers in future.

Or maybe the "B1" registers DO EXIST even on hw4, being accessd by using something like "address of B0 register PLUS some offset", or by using the so-called "BASEBAND_2" and/or "BASEBAND_3" regions - and without mentioning that anywhere in hw4 source code?

DSi Atheros Wifi - Internal I/O - 0xxx00h - RDMA Registers (hw4/hw6)

These registers are same in hw4/hw6, except that:

```
 - base address changed from 30100h (hw4) to 54D00h (hw6)
 - number of regions has increased from 16 (hw4) to 32 (hw6)
 - accordingly, index for STATUS and INT_EN has has changed
```

ATH:030100h/054D00h - DMA_CONFIG ;hw4/hw6

```
  0      DMA_TYPE
  1      RTC_PRIORITY
  2      ENABLE_RETENTION
  3      WLMAC_PWD_EN
  4      WLBB_PWD_EN
  5-31   -
```

ATH:030104h/054D04h - DMA_CONTROL ;hw4/hw6

```
  0      STOP
  1      START
  2-31   -
```

ATH:030108h/054D08h - DMA_SRC ;hw4/hw6
ATH:03010Ch/054D0Ch - DMA_DEST ;hw4/hw6

```
  0-1    -
  2-31   ADDR
```

ATH:030110h/054D10h - DMA_LENGTH ;hw4/hw6

```
  0-11   WORDS
  12-31  -
```

ATH:030114h/054D14h - VMC_BASE ;hw4/hw6

```
  0-1    -
  2-31   ADDR
```

ATH:030118h/054D18h - INDIRECT_REG ;hw4/hw6

```
  0-1    -
  2-31   ID
```

ATH:03011Ch/054D1Ch - INDIRECT_RETURN ;hw4/hw6

```
  0-1    -
  2-31   ADDR
```

ATH:030120h..3015Ch - RDMA_REGION_(0..15)_ - located here in hw4, 16 regions
ATH:054D20h..54D9Ch - RDMA_REGION_(0..31)_ - located here in hw6, 32 regions

```
  0      NEXT
  1      INDI
  2-12   LENGTH
  13-31  ADDR
```

ATH:030160h/054DA0h - DMA_STATUS ;hw4/hw6
ATH:030164h/054DA4h - DMA_INT_EN ;hw4/hw6

```
  0      RUNNING       ;STATUS only (not INT_EN)
  1      STOPPED
  2      DONE
```

```
  3      ERROR
  4-14   ERROR_CODE    ;STATUS only (not INT_EN)
  15-31  -
```

DSi Atheros Wifi - Internal I/O - 03x000h - EFUSE Registers (hw4/hw6)

EFUSE exists in hw4/hw6, with some differences

```
 - base address changed from 31000h (hw4) to 30000h (hw6)
 - the single INTF region (hw4) replaced by two INTF regions (hw6)
 - four new registers added in hw6
 - indices for the two STROBE registers have changed
```

ATH:031000h/030000h - EFUSE_WR_ENABLE_REG ;hw4/hw6

```
  0      V
  1-31   -
```

ATH:031004h/030004h - EFUSE_INT_ENABLE_REG ;hw4/hw6
ATH:031008h/030008h - EFUSE_INT_STATUS_REG ;hw4/hw6

```
  0      V
  1-31   -
```

ATH:03100Ch/03000Ch - BITMASK_WR_REG ;hw4/hw6

```
  0-31   V
```

ATH:031010h/030010h - VDDQ_SETTLE_TIME_REG ;hw4/hw6

```
  0-31   V
```

ATH:031014h/030018h - RD_STROBE_PW_REG ;hw4/hw6

```
  0-31   V
```

ATH:031018h/03001Ch - PG_STROBE_PW_REG ;hw4/hw6

```
  0-31   V
```

ATH:031800h..031FFCh - EFUSE_INTF[0..511] - only one single area in hw4
ATH:030800h..030FFCh - EFUSE_INTF0[0..511] - first area in hw6 here
ATH:031000h..0317FCh - EFUSE_INTF1[0..511] - second area in hw6 here

```
  0-31   R
```

ATH:030014h - VDDQ_HOLD_TIME_REG ;hw6 only

```
  0-31   V
```

ATH:030020h - PGENB_SETUP_HOLD_TIME_REG ;hw6 only

```
  0-31   V
```

ATH:030024h - STROBE_PULSE_INTERVAL_REG ;hw6 only

```
  0-31   V
```

ATH:030028h - CSB_ADDR_LOAD_SETUP_HOLD_REG ;hw6 only

```
  0-31   V
```

DSi Atheros Wifi - Internal I/O - 034000h - More Stuff (hw6)

```
_____________________________________ STEREO _____________________________________
```

ATH:034000h - STEREO0_CONFIG

0-7	POSEDGE
8	MASTER
9	SAMPLE_CNT_CLEAR_TYPE
10	MCK_SEL
11	I2S_WORD_SIZE
12-13	DATA_WORD_SIZE
14-15	STEREO_MONO
16	MIC_WORD_SIZE
17	PCM_SWAP
18	I2S_DELAY
19	RESET
20	MIC_RESET
21	ENABLE
22	REFCLK_SEL
23	SPDIF_ENABLE

ATH:034004h - STEREO0_VOLUME

0-4	CHANNEL0
8-12	CHANNEL1

ATH:034008h - STEREO_MASTER_CLOCK

0	MCK_SEL

ATH:03400Ch - STEREO0_TX_SAMPLE_CNT_LSB
ATH:034010h - STEREO0_TX_SAMPLE_CNT_MSB
ATH:034014h - STEREO0_RX_SAMPLE_CNT_LSB
ATH:034018h - STEREO0_RX_SAMPLE_CNT_MSB

0-15	CH0
16-31	CH1

_____ CHKSUM SEG _____

ATH:035000h - CHKSUM_ACC_DMATX_CONTROL0
ATH:035004h - CHKSUM_ACC_DMATX_CONTROL1
ATH:035008h - CHKSUM_ACC_DMATX_CONTROL2
ATH:03500Ch - CHKSUM_ACC_DMATX_CONTROL3

0	TXEN
1	LITTLEENDIAN

ATH:035010h - CHKSUM_ACC_DMATX_DESC0
ATH:035014h - CHKSUM_ACC_DMATX_DESC1
ATH:035018h - CHKSUM_ACC_DMATX_DESC2
ATH:03501Ch - CHKSUM_ACC_DMATX_DESC3

0-31	ADDR

ATH:035020h - CHKSUM_ACC_DMATX_DESC_STATUS

0	UNDERRUN0
1	UNDERRUN1
2	UNDERRUN2
3	UNDERRUN3
4	BUSERROR
5-8	DESC_INTR
16-23	PKTCNT0
24-25	CHAIN_NUM (R)

ATH:035024h - CHKSUM_ACC_DMATX_ARB_CFG

0	RRMODE
8-13	WGT0

```
  14-19  WGT1
  20-25  WGT2
  26-31  WGT3
```

ATH:035028h - CHKSUM_ACC_RR_PKTCNT01 ;PKT CNT0..1
ATH:03502Ch - CHKSUM_ACC_RR_PKTCNT23 ;PKT CNT2..3

```
  0-8    PKTCNT0 / PKTCNT2   ;9bit each
  16-24  PKTCNT1 / PKTCNT3
```

ATH:035030h - CHKSUM_ACC_TXST_PKTCNT

```
  0-7    N/A ?
  8-15   PKTCNT1             ;8bit each
  16-23  PKTCNT2
  24-31  PKTCNT3
```

ATH:035034h - CHKSUM_ACC_DMARX_CONTROL

```
  0      RXEN
  1      LITTLEENDIAN
```

ATH:035038h - CHKSUM_ACC_DMARX_DESC

```
  0-31   ADDR
```

ATH:03503Ch - CHKSUM_ACC_DMARX_DESC_STATUS

```
  0      OVERFLOW
  1      BUSERROR
  2      DESC_INTR
  16-23  PKTCNT
```

ATH:035040h - CHKSUM_ACC_INTR (R) ;ACC Interrupt (readonly)
ATH:035044h - CHKSUM_ACC_IMASK ;ACC Interrupt Mask

```
  0-3    RX_VAL
  4-16   TX_VAL
```

ATH:035048h - CHKSUM_ACC_ARB_BURST

```
  0-9    MAX_TX
  10     INCR16_EN
  11     INCR8_EN
  16-25  MAX_RX
```

ATH:035050h - CHKSUM_ACC_RESET_DMA

```
  0      TX
  1      RX
```

ATH:035054h - CHKSUM_CONFIG

```
  0      CHKSUM_SWAP
  4-9    TXFIFO_MAX_TH
  16-21  TXFIFO_MIN_TH
  22-31  SPARE
```

_____ MMAC _____

ATH:038000h - RX_FRAME0
ATH:038008h - RX_FRAME1

```
  0      OWN
  1-12   LEN                        (R)
  13-14  SEQ_NUM                    (R)
```

ATH:038004h - RX_FRAME_0
ATH:03800Ch - RX_FRAME_1

0-31 ADDR

ATH:038010h - MMAC_INTERRUPT_RAW
ATH:038014h - MMAC_INTERRUPT_EN

0 RX_DONE0
1 RX_CRC_FAIL0
2 ACK_RESP_FAIL0
3 RX_DONE1
4 RX_CRC_FAIL1
5 ACK_RESP_FAIL1
6 RX_ERR_OVERFLOW
7 TX_DONE
8 TX_DONE_ACK_MISSING
9 TX_DONE_ACK_RECEIVED
10 TX_ERROR

ATH:038018h - RX_PARAM1

0-31 VAP_ADDR_L

ATH:03801Ch - RX_PARAM0

0-15 VAP_ADDR_U
16-21 SIFS
22-23 CAPTURE_MODE
24-26 TYPE_FILTER
27 LIVE_MODE

ATH:038020h - TX_COMMAND0

0-11 LEN
12 CRC
13 EXP_ACK

ATH:038024h - TX_COMMAND

0-31 ADDR

ATH:038028h - TX_PARAM

0 ACK_MODE_EN
1-6 ACK_TIMEOUT
7-14 BACKOFF
15 FORCE_ACKF_RSSI
16-23 ACKF_RSSI

ATH:03802Ch - BEACON_PARAM

0-15 INTERVAL
16-27 LEN
28 EN
29 CRC
30 RESET_TS

ATH:038030h - BEACON

0-31 ADDR

ATH:038034h - TSF_L
ATH:038038h - TSF_U

0-31 COUNT

FPGA

ATH:039000h - FPGA_REG1

2 DCM_RELEASE
4-7 EMUL_RADIO_CLOCK_RATIO

```
  8-9    LONG_SHIFT_CHAIN_OVERRIDE_INDEX
  10     ENABLE_LONG_SHIFT_CHAIN_OVERRIDE_INDEX
  11-15  LONG_SHIFT_DRIVE_PHASE
  16-20  LONG_SHIFT_SAMPLE_PHASE
  21-30  SPARE_FPGA_REG1
  31     FPGA_SRIF_DELAY
```

ATH:039004h - FPGA_REG2

```
  0-3    FPGA_PLATFORM_TYPE
  4-7    FPGA_IP_RELEASE_VERSION
  8-11   FPGA_IP_REVISION
  12     FPGA_OWL_PLL_ENABLED
  13     FPGA_LOOPBACK_I2C
  14-31  FPGA_SPARE
```

ATH:039008h - FPGA_REG4

```
  0      RADIO_0_TCK
  1      RADIO_0_TDI
  2      RADIO_0_TMS
  3      RADIO_0_TDO
```

_____ BRIDGE INTR _____

ATH:040000h - INTERRUPT
ATH:040004h - INTERRUPT_MASK

```
  0-7    RX_(0..7)_COMPLETE
  8-15   RX_(0..7)_END
  16-23  TX_(0..7)_COMPLETE
  24-31  TX_(0..7)_END
```

_____ MII _____

ATH:040100h - MII0_CNTL

```
  0-1    SELECT
  2      MASTER
  4-5    SPEED
  8-9    RGMII_DELAY
```

ATH:040104h - STAT_CNTL (whatever, MII related?)

```
  0      AUTOZ
  1      CLRCNT
  2      STEN
  3      GIG
```

_____ MDIO _____

ATH:040200h - MDIO_REG[0..7]

```
  0-15   VALUE
```

ATH:040220h - MDIO_ISR

```
  0-7    REGS
  8-15   MASK
```

ATH:040224h - PHY_ADDR (whatever, MDIO related?)

```
  0-2    VAL
```

_____ BRIDGE RX/TX _____

ATH:040800h - GMAC_RX_0_DESC_START_ADDRESS
ATH:040C00h - GMAC_TX_0_DESC_START_ADDRESS

```
  0-31   ADDRESS
```

ATH:040804h - GMAC_RX_0_DMA_START
ATH:040C04h - GMAC_TX_0_DMA_START

```
  0      START
  4      RESTART
```

ATH:040C08h - GMAC_TX_0_INTERRUPT_LIMIT

```
  0-3    COUNT
  4-15   TIMEOUT
```

ATH:040808h - GMAC_RX_0_BURST_SIZE
ATH:040C0Ch - GMAC_TX_0_BURST_SIZE

```
  0-1    BURST
```

ATH:04080Ch - GMAC_RX_0_PKT_OFFSET

```
  0-7    OFFSET
```

ATH:040810h - GMAC_RX_0_CHECKSUM

```
  0      TCP
  1      UDP
```

ATH:040814h - GMAC_RX_0_DBG_RX

```
  0-3    STATE
```

ATH:040C10h - GMAC_TX_0_DBG_TX

```
  16-31  FIFO_TOTAL_LEN
  0-2    STATE
```

ATH:040818h - GMAC_RX_0_DBG_RX_CUR_ADDR
ATH:040C14h - GMAC_TX_0_DBG_TX_CUR_ADDR

```
  0-31   ADDR
```

ATH:04081Ch - GMAC_RX_0_DATA_SWAP
ATH:040C18h - GMAC_TX_0_DATA_SWAP

```
  0      SWAP
  1      SWAPD
```

_____ USB CAST _____

ATH:054000h - ENDP0

```
  0-7    MAXP
  16     STALL
  17     HSNAK
  20     DSTALL
  23     CHGSETUP
```

ATH:054008h - OUT1ENDP
ATH:054010h - OUT2ENDP
ATH:054018h - OUT3ENDP
ATH:054020h - OUT4ENDP
ATH:054028h - OUT5ENDP
ATH:05400Ch - IN1ENDP
ATH:054014h - IN2ENDP
ATH:05401Ch - IN3ENDP
ATH:054024h - IN4ENDP
ATH:05402Ch - IN5ENDP

```
  0-10   MAXP
```

```
  18-19  TYPE
  20-21  ISOD
  22     STALL
  23     VAL
  24     ISOERR
  28     HCSET          ;<-- for INxENDP registers only (not OUTxENDP)
```

ATH:05408Ch - USBMODESTATUS

```
  0      LS
  1      FS
  2      HS
  4      HOST
  5      DEVICE
```

ATH:054188h - EPIRQ ;Endpoint Interrupt Request
ATH:054194h - EPIEN ;Endpoint Interrupt Enable

```
  0-15   IN 0..15
  16-31  OUT 0..15
```

ATH:05418Ch - USBIRQ ;USB Interrupt Request
ATH:054198h - PIEN ;P Interrupt Enable

```
  0      SUDAV          IR
  1      SOF            IR
  2      SUTOK          IR
  3      SUSP           IR
  4      URES           IR
  5      HSPEED         IR
  6      OVERFLOW       IR
  7      LPM            IR
  16-31  OUTP           NGIRQ ?
```

ATH:0541A4h - FNCTRL

```
  0-2    MFR
  3-7    FRMNR0
  8-13   FRMNR1
  16-22  FNADDR
  24-31  CLKGATE
```

ATH:0541BCh - OTGREG

```
  0      OTGIRQ_IDLEIRQ
  1      OTGIRQ_SRPDETIRQ
  2      OTGIRQ_LOCSOFIRQ
  3      OTGIRQ_VBUSERRIRQ
  4      OTGIRQ_PERIPHIRQ
  8-11   OTGSTATE
  16     OTGCTRL_BUSREQ
  17     OTGCTRL_ABUSDROP
  18     OTGCTRL_ASETBHNPEN
  19     OTGCTRL_BHNPEN
  20     OTGCTRL_SRPVBUSDETEN
  21     OTGCTRL_SRPDATDETEN
  23     OTGCTRL_FORCEBCONN
  24     OTGSTATUS_BSE0SRP
  25     OTGSTATUS_CONN
  27     OTGSTATUS_ASESSVAL
  28     OTGSTATUS_BSESSEND
  29     OTGSTATUS_AVBUSVAL
  30     OTGSTATUS_ID
```

ATH:0541CCh - DMASTART
ATH:0541D0h - DMASTOP

```
  0-15   IN 0..15
```

```
  16..31 OUT 0..15
```

ATH:054400h - EP0DMAADDR ;what is Endpoint 0, normal are 1..5, not 0?
ATH:054420h - EP1DMAADDR
ATH:054440h - EP2DMAADDR
ATH:054460h - EP3DMAADDR
ATH:054480h - EP4DMAADDR
ATH:0544A0h - EP5DMAADDR

```
  2-31   ADDR
```

ATH:05442Ch - OUT1DMACTRL
ATH:05444Ch - OUT2DMACTRL
ATH:05446Ch - OUT3DMACTRL
ATH:05448Ch - OUT4DMACTRL
ATH:0544ACh - OUT5DMACTRL

```
  2-15   RINGSIZ
  16     ENDIAN
  17     DMASTOP
  18     DMASTART
  20     DMATUNLIM
  21     DMANINCR
  22     DMARING
  25     HLOCK
  26-27  HSIZE
  28-31  HRPROT
```

ATH:0D8000h - USB_IP_BASE

```
  ?

 _____________________________ I2C SLAVE _____________________________
```

ATH:054E00h - I2CFIFOCONTROL ;FIFO Control

```
  0      FIFO RESET
  1      FIFO PREFETCH
  2-4    FIFO READ LENGTH
  5-14   FIFO READ THRESHOLD
  15     FIFO READ STALL
  16-18  FIFO WRITE LENGTH
  19-28  FIFO WRITE THRESHOLD
  29     FIFO WRITE STALL
```

ATH:054E04h - I2CFIFOREADPTR ;FIFO Read WR+RD Pointers
ATH:054E10h - I2CFIFOWRITEPTR ;FIFO Write WR+RD Pointers

```
  0-9    WR PTR                      (R)
  16-25  RD PTR                      (R)
```

ATH:054E08h - I2CFIFOREADUPDATE ;FIFO Read Update
ATH:054E14h - I2CFIFOWRITEUPDATE ;FIFO Write Update

```
  0-10   UPDATE
```

ATH:054E0Ch - I2CFIFOREADBASEADDR ;FIFO Read Base Address
ATH:054E18h - I2CFIFOWRITEBASEADDR ;FIFO Write Base Address

```
  0-31   BASE
```

ATH:054E1Ch - I2CMEMCONTROL ;Mem Control

```
  0      RESET
  1      FLUSH
```

ATH:054E20h - I2CMEMBASEADDR ;Mem Base Address

```
  0-31   BASE
```

ATH:054E24h - I2CREGREADDATA ;Reg Read Data
ATH:054E28h - I2CREGWRITEDATA ;Reg Write Data (R) ;uh, write is read-only?
```
  0-31   DATA
```

ATH:054E2Ch - I2CREGCONTROL ;Reg Control
```
  0      RESET
  1      READ STALL
  2      WRITE STALL
  3-5    READ COUNT              (R)
  6-8    WRITE COUNT             (R)
  9      READ EMPTY              (R)
  10     WRITE FULL              (R)
```

ATH:054E30h - I2CCSRREADDATA ;Csr Read Data
ATH:054E34h - I2CCSRWRITEDATA ;Csr Write Data (R) ;uh, write is read-only?
```
  0-5    DATA (6bit, what is that?)
```

ATH:054E38h - I2CCSRCONTROL ;Csr Control
```
  0-7    READDELAY
  8      CLOCKREQUESTENABLE
  9-11   FILTERCLOCKSELECT
  12-14  FILTERCLOCKSCALE
  15-17  FILTERSDARXSELECT
  18-20  FILTERSCLRXSELECT
```

ATH:054E3Ch - I2CFILTERSIZE ;Filter Size
```
  0-7    SDA RX SIZE
  8-15   SCL RX SIZE
```

ATH:054E40h - I2CADDR ;Address
```
  0-6    FIFO ADDR
  8-14   MEM ADDR
  16-22  REG ADDR
  24-30  CSR ADDR
```

ATH:054E44h - I2CINT ;Interrupt Status
ATH:054E48h - I2CINTEN ;Interrupt Enable
```
  0      FIFO READ START INT           ;\
  1      FIFO READ FINISH INT          ;
  2      FIFO WRITE START INT          ;
  3      FIFO WRITE FINISH INT         ; R/W
  4      REG READ START INT            ;
  5      REG READ FINISH INT           ;
  6      REG WRITE START INT           ;
  7      REG WRITE FINISH INT          ;/
  8      FIFO READ EMPTY INT           ;\
  9      FIFO WRITE FULL INT           ; For Status: R
  10     FIFO READ THRESHOLD INT       ; For Enable: R/W
  11     FIFO WRITE THRESHOLD INT      ;/
  12     CSR INT                       ;-R/W
```

ATH:054E4Ch - I2CINTCSR ;Int Csr
```
  0      INT    ;Status          (R)
  1      INTEN  ;Enable
```

```
_____________________________________ MAP I2S _____________________________________
```

ATH:055000h - MBOX_FIFO

```
  0-19   DATA
```

ATH:055004h - MBOX_FIFO_STATUS
```
  0      FULL
  2      EMPTY
```

ATH:055008h - MBOX_DMA_POLICY
```
  0      RX_ORDER
  1      RX_QUANTUM
  2      TX_ORDER
  3      TX_QUANTUM
  4-7    TX_FIFO_THRESH0
```

ATH:05500Ch - MBOX0_DMA_RX_DESCRIPTOR_BASE
ATH:055014h - MBOX0_DMA_TX_DESCRIPTOR_BASE
```
  2-27   ADDRESS
```

ATH:055010h - MBOX0_DMA_RX_CONTROL
ATH:055018h - MBOX0_DMA_TX_CONTROL
```
  0      STOP
  1      START
  2      RESUME
```

ATH:05501Ch - MBOX_FRAME
```
  0      RX_SOM
  2      RX_EOM
```

ATH:055020h - FIFO_TIMEOUT
```
  0-7    VALUE
  8      ENABLE
```

ATH:055024h - MBOX_INT_STATUS
ATH:055028h - MBOX_INT_ENABLE
```
  0      RX_NOT_FULL
  2      TX_NOT_EMPTY
  4      RX_UNDERFLOW
  5      TX_OVERFLOW
  6      TX_DMA_COMPLETE
  8      TX_DMA_EOM_COMPLETE
  10     RX_DMA_COMPLETE
```

ATH:05502Ch - MBOX_FIFO_RESET
```
  0      TX_INIT
  2      RX_INIT
```

ATH:055030h - MBOX_DEBUG_CHAIN0
ATH:055034h - MBOX_DEBUG_CHAIN1
```
  0-31   ADDRESS
```

ATH:055038h - MBOX_DEBUG_CHAIN0_SIGNALS
ATH:05503Ch - MBOX_DEBUG_CHAIN1_SIGNALS
```
  0-31   COLLECTION
```

_____ MAP RF _____

ATH:xxx400h - INT_PENDING[0..11]
```
  0-31   REG (32bit x 12 entries)
```

ATH:xxx460h - INT_SRC (R)

```
  0-11   REG (12bit)    (R)
```

ATH:xxx430h - BB_WR_MASK_0
ATH:xxx434h - BB_WR_MASK_1
ATH:xxx438h - BB_WR_MASK_2
ATH:xxx43Ch - BB_WR_MASK_3
ATH:xxx448h - BB_RD_MASK_0
ATH:xxx44Ch - BB_RD_MASK_1
ATH:xxx450h - BB_RD_MASK_2
ATH:xxx454h - BB_RD_MASK_3

```
  0-10   REG (11bit)
```

ATH:xxx440h - RF_WR_MASK_0
ATH:xxx444h - RF_WR_MASK_1
ATH:xxx458h - RF_RD_MASK_0
ATH:xxx45Ch - RF_RD_MASK_1

```
  0-8    REG (9bit)
```

ATH:xxx000h - RAM1[0..255]

```
  0-7    DATA (8bit)
  8-31   -
```

ATH:xxx800h - RAM2[0..127]

```
  0-6    DATA (7bit)
  7-31   -
```

```
_____________________________________ ODIN _____________________________________
```

ATH:xxx000h - PHY_CTRL0

```
  0-2    PLL_ICP
  3-5    PLL_RS
  6-14   PLL_DIV
  15-17  PLL_MOD
  18     PLL_OVERIDE
  19     TEST_SPEED_SELECT
  20     RX_PATTERN_EN
  21     TX_PATTERN_EN
  22     ANA_LOOPBACK_EN
  23     DIG_LOOPBACK_EN
  24-31  LOOPBACK_ERR_CNT   (R)
```

ATH:xxx004h - PHY_CTRL1

```
  0-1    RX_FILBW_SEL
  2      RX_FORCERXON
  3      RX_BYPASSEQ
  4      RX_LOWR_PDET
  5-6    RX_SELIR_100M
  7      RX_SELVREF0P6
  8      RX_SELVREF0P25
  9-11   RX_RSVD
  12     NO_PLL_PWD
  13     FORCE_SUSPEND
  18-19  TX_PATTERN_SEL
  20     USE_PLL_LOCKDETECT
  21-22  USE_PLL_LOCK_DLY_SEL
  23-25  CLKOBS_SEL
  26     ENABLE_REFCLK_GATE
  27     DISABLE_CLK_GATING
  31     PLL_OBS_MODE_N
```

ATH:xxx008h - PHY_CTRL2

```
0      HSTXBIAS_PS_EN
1      HSRXPHASE_PS_EN
2-7    PWD_IPLL
8-13   PWD_ISP
20     TX_CAL_EN
21     TX_CAL_SEL
22-25  TX_MAN_CAL
26     TX_LCKDET_OVR
27-30  TX_RSVD
31     PWD_EXTBIAS
```

ATH:xxx00Ch - PHY_CTRL3

```
0-18   PWD_ITX
21     TX_DISABLE_SHORT_DET
22-24  TX_SELTEST
25     TX_STARTCAL
```

ATH:xxx010h - PHY_CTRL4

```
0-11   PWD_IRX
```

ATH:xxx014h - PHY_CTRL5

```
0-6    TX_BIAS_DELAY
7-12   EB_WATERMARK
13     FORCE_IDDQ
14     FORCE_TEST_J
15     FORCE_TEST_K
16     FORCE_TEST_SE0_NAK
17     TEST_JK_OVERRIDE
18-19  XCVR_SEL
20     TERM_SEL
21     SUSPEND_N
22     DP_PULLDOWN
23     DM_PULLDOWN
24     HOST_DISCON_FIX_ON
25     HOST_DISCON_DETECT_ON
26-28  HOST_DISCON_SAMPLE_WIDTH
```

ATH:xxx018h - PHY_CTRL6

```
0      AVALID
1      BVALID
2      VBUSVALID
3      SESSEND
4      IDDIG
```

ATH:xxx01Ch - PHY_STATUS (R)

```
0-3    TX_CAL      (R)
```

DSi GPIO Registers

4004C00h DSi7 - GPIO Data In (R) (even in DS mode)

```
0     GPIO18[0]   ;\maybe 1.8V signals? (1=normal)
1     GPIO18[1]   ; (maybe these are the three "NC" pins on CPU,
2     GPIO18[2]   ;/near to the other GPIO pins)
3     Unused (0)
4     GPIO33[0] Probably "GPIO330" test point on mainboard
5     GPIO33[1] Headphone connect (HP#SP) (0=None, 1=Connected)
6     GPIO33[2] Powerbutton interrupt (0=Short Keydown Pulse, 1=Normal)
7     GPIO33[3] sound enable output (ie. not a useful input)
```

One of the above is probably the "IRQ_O" signal on mainboard (possibly the "power button" bit; if so, then that bit might be some general interrupt from the "BPTWL" chip, rather than being solely related to the power button).
Bit0-2 might be unused "NC" pins. Bit4 might be GPIO330, which might be just a test point without other connection. Bit7 might be connected to one of unknown vias (maybe that near GPIO330 test point?), and which might connect to somewhere (probably to TSC/sound as the bit seems to enable sound output - though, the cooking coach cart sets interrupt edge select bit7; which hints that the pin could be used for something; the interrupt isn't actually enabled though).
Some bits seem to be floating high-z (when switching from output/low to input they won't <instantly> get high).

4004C00h DSi7 - GPIO Data Out (W)

```
0-2   GPIO18[0-2] Data Output (0=Low, 1=High)
3     Unused (0)
4-7   GPIO33[0-3] Data Output (0=Low, 1=High)
```

Used only when below is set to direction=out. Should be 80h (sound enable).

4004C01h DSi7 - GPIO Data Direction (R/W)

```
0-2   GPIO18[0-2] Data Direction (0=Normal/Input, 1=Output)
3     Unused (0)
4-7   GPIO33[0-3] Data Direction (0=Normal/Input, 1=Output)
```

Should be usually set to 80h (bit0-6=Input, bit7=Output). When using output direction, the "Data In" register will return the "Data Out" value ANDed with external inputs.

Observe that HP#SP could be used as Output (output/low will probably cause the DSi to believe that there is no headphone connected, thus forcing the internal speakers to be enabled; possibly with/without disabling the headphones?).

4004C02h DSi7 - GPIO Interrupt Edge Select (R/W)

```
0-2   GPIO18[0-2] Interrupt Edge Select (0=Falling, 1=Rising)
3     Unused (0)
4-7   GPIO33[0-3] Interrupt Edge Select (0=Falling, 1=Rising)
```

4004C03h DSi7 - GPIO Interrupt Enable (R/W)

```
0-2   GPIO18[0-2] Interrupt Enable (0=Disable, 1=Enable)
3     Unused (0)
4-7   GPIO33[0-3] Interrupt Enable (0=Disable, 1=Enable)
```

4004C04h - DSi7 - GPIO_WIFI (R/W)

```
0     Unknown (firmware keeps this bit unchanged when writing)
1-7   Zero
8     Wifi Mode (0=New Atheros/DSi-Wifi mode, 1=Old NDS-Wifi mode)
9-15  Zero
```

Bit8 must be properly configured for DWM-W024 wifi boards (must be 1 for NDS-Wifi mode, and must be 0 for DSi-Wifi; else SDIO Function 1 commands won't work).
DWM-W015 boards seem to work fine regardless of that bit (unknown if the bit does have any (minor) effects on that boards at all).

DSi Console IDs

The DSi contains several unique per-console numbers:

```
CPU/Console ID     - Found in Port 4004D00h, in AES Keys, and in Files
eMMC CID Register  - Found in Main RAM, and in eMMC CID register
Serial/Barcode     - Found in Main RAM, and on stickers, and in HWINFO_S.dat
Wifi MAC Address   - Found in Main RAM, and in Wifi FLASH
Nintendo WFC ID    - Found in Wifi FLASH
```

4004D00h - DSi7 - CPU/Console ID Code (64bit) (R)

```
  0-63  CPU/Console ID Code
```

This appears to be a PROM inside of the CPU TWL chip. The value is used to initialize KEY_X values for eMMC encryption/decryption. Common 64bit settings are:

```
  08201nnnnnnnn1nnh  for DSi (EUR) and DSi XL (USA)
  08202nnnnnnnn1nnh  for DSi XL
  08203nnnnnnnn1nnh  for DSi XL
  08204nnnnnnnn1nnh  for DSi
  08A15??????????h  for DSi
  08A18nnnnnnnn1nnh  for DSi (USA) (black)
  08A19nnnnnnnn1nnh  for DSi (USA)
  08A20nnnnnnnn1nnh  for DSi
  08A21nnnnnnnn1nnh  for DSi
  08A22??????????h  for DSi (USA)
  08A23??????????h  for DSi (EUR)
  08C267B7xxxxxxxxh  for 3DS
  6B27D20002000000h  for n3DS (unknown how that has happened)
```

The "n" digits appear to be always in BCD range (0..9), the other digits appear to be fixed (on known consoles). The 64bit value is also stored as 16-byte ASCII string in "dev.kp". And, the ASCII string is also stored in footer of "Tad Files on SD Cards".

Port 4004D00h should be read only if the flag in 4004D08h is zero. Moreover, Port 4004D00h can be read only by firmware, and get's disabled for all known games, so most exploits will only see zeroes in 4004D00h..4004D08h.

Easiest way to obtain the 64bit value would be extracting it from SD Card (using modified "DSi SRL Extract" source code).

DSi SD/MMC DSiware Files on External SD Card (.bin aka Tad Files)

Obtaining the 64bit value by DSi software is working only indirectly:

With sudokuhax it can be simply "read" from 40044E0h (LSW) and 40044ECh (MSW). Whereas, that ports are write-only, so it needs some small efforts to "read" them.

With DSi cartridge exploits it's a bit more difficult: The values at 40044D4h..40044FBh are destroyed, but 40044D0h..40044D3h is left intact, which can be used to compute the original MSW value at 40044ECh, using a bunch of constants and maths operations (caution: the result may depend on carry-in from unknown LSBs, eg. the MSW may appear as 08A2nnnnh or 08E2nnnnh). Next, one can simply brute-force the LSW (there should be only 10 million combinations (assuming it to be a BCD number with one fixed digit), which could be scanned within less than 6 minutes using the DSi AES hardware).

Note: JimmyZ made some PC tools ("TWLbf" and "bfCL") that can bruteforce the Console ID or CID (or both) from encrypted eMMC images.

4004D08h - DSi7 - CPU/Console ID Flag (1bit) (R)

```
  0     CPU/Console ID Flag (0=Okay/Ready, 1=Bad/Busy)
  1-15  Unknown/Unused (0)
```

Some flag that indicates whether one can read the CPU/Console ID from Port 4004D00h. The flag should be usually zero (unknown when it could be nonzero, maybe shortly after power-up, or maybe when the internal PROM wasn't programmed yet; which should never happen in retail units).

eMMC CID Register

The CID can be read via SD/MMC commands, and it's also stored at 2FFD7BCh in RAM; the RAM value is little-endian 120bit (ie. without the CRC7 byte), zeropadded to 16-bytes (with 00h in MSB); the value looks as so;

```
  MY ss ss ss ss 03 4D 30 30 46 50 41 00 00 15 00  ;DSi Samsung KMAPF0000M-S998
  MY ss ss ss ss 32 57 37 31 36 35 4D 00 01 15 00  ;DSi Samsumg KLM5617EFW-B301
  MY ss ss ss ss 30 36 35 32 43 4D 4D 4E 01 FE 00  ;DSi ST NAND02GAH0LZC5 rev30
  MY ss ss ss ss 31 36 35 32 43 4D 4D 4E 01 FE 00  ;DSi ST NAND02GAH0LZC5 rev31
  MY ss ss ss ss 03 47 31 30 43 4D 4D 00 01 11 00  ;3DS whatever chiptype?
  MY ss ss ss ss 07 43 59 31 47 34 4D 00 01 15 00  ;3DS Samsung KLM4G1YE0C-B301
```

The value is used to initialize AES_IV register for eMMC encryption/decryption.

The "MY" byte contains month/year; with Y=0Bh..0Dh for 2008..2010 (Y=0Eh..0Fh would be 2011..2012, but

there aren't any known DSi/3DS consoles using that values) (unknown how 2013 and up would be assigned; JEDEC didn't seem to mind to define them yet). The "ss" digits are a 32bit serial number (or actually it looks more like a 32bit random number, not like an incrementing serial value).
Without a working exploit (for reading RAM at 2FFD7BCh), the CID could be obtained by connecting wires to the eMMC chip. However, this might require whatever custom hardware/software setup (unknown if any standard tools like PC card readers are able to read the CID value).
Note: JimmyZ made some PC tools ("TWLbf" and "bfCL") that can bruteforce the Console ID or CID (or both) using hex values extracted from encrypted eMMC images.
To speedup CID bruting, one can extract the MY datecode from "Samsung YWW, KMAPF0000M-S998" text printed on the eMMC chip: the "YWW" is year/week, eg. 8xx means 2008, and translates to year "B" in the "MY" field.
For the ST chips, there seems to me a similar year/week (reading "KOR 99 YWW", but the year/week on ST chips is usually about a month older than the month/year in CID). Known ST date codes are AC/BC (Oct2009/Nov2009) for the "rev0" ones, and CC/1D (Dec2009/Jan2010) for the "rev1" ones (ie. the ST chips seem to have been mostly used in early DSi XL models).

Serial/Barcode
The barcode is found on a sticker on the bottom of console (and on an identical sticker underneath of the battery). It's also stored as ASCII string in HWINFO_S.dat file, and at 2FFFD71h in RAM.
The serial/barcode consists of 2-3 letters, followed by 8 digits, followed by a checksum digit.
The first letter indicates the console model:

```
DSi             T  (or V for devunits)
DSi XL/LL       W  (or unknown for devunits)
3DS             C  (or E for devunits)
3DS XL/LL       S  (or R for devunits)
2DS             A  (or P for devunits)
New 3DS         Y  (or Yxx00 for devunits)
New 3DS XL/LL   Q  (or Qxx00 for devunits)
New 2DS XL/LL   N  (or Nxx01 for devunits)
```

The next 1-2 letter(s) indicate the region:

```
JPN Japan                              JF,JH,JM
USA North America                      W
USA Middle East, Southeast Asia        S
EUR Europe                             EF,EH,EM
AUS Australia (for 3DS: part of EUR)   AG,AH
CHN China (iQue)                       CF,CH,CM
KOR South Korea                        KF,KH,KM
```

The checksum can be calculated across 1st-8th digit (and ignoring the leading letters):

```
9th = (250 - (1st+3rd+5th+7th) - 3*(2nd+4th+6th+8th)) mod 10
```

Unknown if the barcode is internally used for any purposes (such like encryption, or network identification).

dev.kp
This file contains another "TWxxxxxxxx" ID (with "xxxxxxxx" being a 32bit lowercase hex number), this 32bit Console ID is also used in .tik files (except in tickets for free system titles).

Wifi MAC Address - Found in Main RAM, Wifi FLASH, and Wifi EEPROM
The MAC is a unique 48bit number needed for Wifi communications. The MAC is stored in SPI bus Wifi FLASH, and it's also stored at 2FFFCF4h in RAM. The same MAC value is also stored in I2C bus Wifi EEPROM. The MAC can be also viewed in Firmware (see System Settings, Internet, Options, System Information). Common values for DSi are:

```
00 22 4C xx xx xx    ;seen in DSi XL
00 23 CC xx xx xx    ;seen in DSi
00 24 1E xx xx xx    ;seen in DSi
00 27 09 xx xx xx    ;seen in DSi
```

The value isn't used for eMMC encryption (the eMMC is still accessible when swapping the Wifi daughterboard). However, the MAC value is included in game ".bin" files stored on SD card (unknown if that is causing any issues when loading those games on a console with swapped Wifi daughterboard).

Nintendo WFC ID
This is some unknown purpose value stored in Wifi FLASH. The value can be viewed in Firmware (see System Settings, Internet, Options, System Information). The firmware does only show the lower 43bit of the value, in decimal format, multiplied by 1000, whilst the actual WFC in FLASH seems to be about 14 bytes (112bit). The firmware does also allow to "delete" and "transfer" the "WFC Configuration" (whatever that means).

Flipnote Studio ID
This ID consists of a 16-digit HEX number (can be viewed by clicking the "tool" symbol in upper right of Flipnote's main menu).
The first 8-digits have unknown meaning. The last 8-digits are same as the last 8-digits of the wifi MAC address.

DSi Unknown Registers

40021A0h ... second NDS cart slot, DSi prototype relict (?)
```
ARM9: Can be set to 00000000h or 0000E043h
ARM7: Can be set to 00000000h or 0000E043h
```

40021A4h ... second NDS cart slot, DSi prototype relict (?)
```
ARM9: Can be set to 20000000h or FF7F7FFFh
ARM7: Can be set to 00000000h or FF7F7FFFh
```
Can be all-zero on arm7, uh, or is that due to port being disabled somehow?

DSi Notes

DSi Detection
Cartridges are using the same executable for NDS and DSi mode, the executable must thus detect whether it is running on a NDS console or DSi console. At ARM9 this is usually done as follows:
```
if ([4004000h] AND 03h)=01h then DSi_mode else NDS_mode
```
On ARM7 side, the executables are attempting to do the same thing, but they are (maybe accidently) skipping the detection depending on a 2nd I/O port:
```
;Caution: Below detection won't work with DSi exploits (because they are
;  usually having the ARM7 SCFG registers disabled - it would be thus better
;  to do the dection only on ARM9 side as described above, and then forward
;  the result to ARM7 side).
if ([4004008h] AND 80000000h)=0 then skip_detection_and_assume_NDS_mode
else if ([4004000h] AND 03h)=01h then DSi_mode else NDS_mode
```
In DSi_mode, the game can use additional hardware features, and it must be also aware of some changed BIOS SWI functions.
In NDS_mode, the game can use only old hardware features, in case of DSi-exclusive games: The cartridge must contain a small NDS function that displays a message saying that the game will work only on DSi consoles.

DSi Executables
The boot executables & entrypoints are always defined in the ARM9/ARM7 entries in cartridge header, regardless of whether the game is running on a NDS console or DSi console. The ARM9/ARM7 executables are thus restricted to max 2MByte size (for not violating the NDS memory limit). The ARM9i/ARM7i entries are allowing to load additional code or data, presumably to addresses (almost) anywhere within the DSi's 16MByte memory space. Of course, if you want to use separate executables for NDS and DSi mode, then you can put some small bootstrap loader into the ARM9/ARM7 area, which will then load the actual main executables.

DSi Official Games
There are only a few DSi-Exclusive games, and quite a lot of DSi-Enhanced games. The package of that games doesn't seem bear any DSi-logos, so it's hard to tell if a game contains DSi features (except via inofficial

databases).

DSi Homebrew Games
These are extremly rare, hard to find, and practically non-existant yet. Most webpages are mis-offering NDS games as "homebrew DSi games" (which, well, they may work on DSi, but only in NDS mode).
There are a few "real" DSi homebrew games/emus/demos: Atari 5200 EMU, Atari 7800 EMU, CQuake, DSx86, GBA Emulator, Project Legends DS, Sandbox Engine, StellaDS, Zoomx3 - most or all of them seem to require weird DSi exploits (probably CycloDS iEvolution flashcarts), the games appear to be completely lacking DSi cartridge headers, without even setting the DSi flag in cartheader[012h].bit1 (and instead they do contain pre-historic NDS-passme headers for booting from GBA slot, which is definetely incompatible with real DSi consoles).

DSi Exploits

DSi executables require RSA signatures (signed with Nintendo's private key), which is making it impossible to run any unlicensed/homebrew code, except via exploits (eg. modified .sav data for tweaking licensed games into booting unlicensed code).

Unlaunch.dsi - first ever released DSi bootcode exploit
This exploit allows to get control of the DSi almost immediately after power-up with full SCFG_EXT access rights. That's making it the best possible exploit for any purpose.
It can be installed via any DSiware exploits (eg. Flipnote), or via hardmod.

Ready for use DSiware Exploits (crashing system titles from SD card)
There are two exploitable titles that can be exploited via simple SD card files (without needing hardmod or downgrading): DSi Camera and Flipnote
DSi Camera (aka Memory Pit exploit):
The DSI Camera is installed in all regions, and people can't install it, so this exploit should work for everyone.
The original Memory Pit exploit (from shutterbug2000) worked only on unspecified firmwares, but there is newer Memory Pit (from zoogie) the works in all regions and firmware versions.
Usage: Replace the SD:\private\ds\app\484E494A\pit.bin file, then go to Camera SD Card Album.
Flipnote:
Flipnote exists for US+EUR+AUS+JAP regions (not CHN or KOR), and it has been available for free (either pre-installed, or free download from DSi shop), so many people should have that title (though, surprisingly, quite some people don't have it because they had never downloaded it for free when the DSi shop was still online; and some might have deleted it).
Anyways, if you have flipnote: Use the "Flipnote Lenny (or whatever it is called)" exploit, working for all four flipnote regions (US+EUR+AUS+JAP), the download URL seems to be this weird link:

```
https://davejmurphy.com/%25CD%25A1-%25CD%259C%25CA%2596-%25CD%25A1/
```

downloading seems to require buying a newer PC, and a browser with strong https support and youtube playing capabilities for watching the installation guide (unless the download contains instructions in regular .txt format).

Other DSiware Exploits (DSi shop downloads crashed with .sav files)
Installing DSiware exploits requires downgrading the firmware (and in case of sudoku: also downgrading the game). Downgrading can be done by soldering a few wires to the eMMC chip and then using utilities like "TWLTool". Most recommended title would be Sudoku (it's cheapest and doesn't require any menu navigation; apart from getting through a "touch to start" screen).

```
Region       Price Title
US,EU/AU     $2    Sudoku (Electronic Arts)   (updated version in DSi shop)
US,EU/AU     $5    Fieldrunners        (original version still in DSi shop)
US,EU/AU     $5    Guitar Rock Tour ("grtpwn")      (no longer in DSi shop)
US,EU/AU,JP  $8    Legends of Exidia   (original version still in DSi shop)
US,...?      Free  Zelda 4 Swords                   (no longer in DSi shop)
```

Once when the game+exploit are installed, the exploits will boot a file called "boot.nds" from SD card (eg. rename dslink to that name for wifi boot).
The DSi shop closed in 2017, so one can no longer legally buy DSiware. One could install pirate copies with tweaked .tik files for exploitable games, but it would be pointless (doing so would require a hardmod, and if you have a hardmod, then you could as well install unlaunch directly).
Note: Originally, dsiwarehax were installed via SD card without soldering, but that works only when having downloaded the exploitable games prior to release of firmware 1.4.2.
As of 2017, the DSi shop is closing (one cannot add new DSi points to the shop account, but can still use old DSi points that are already on the account for a few months).
Note: Buying DSiware was quite expensive (ie. buying a $15 wii/dsi point card worked, but you couldn't resell or reuse the remaining dsi points on other consoles).

DSi Cartridge Exploits
Exploitable DSi cartridges have been mass-produced as gifts for fat, senile or unhealthy family members, and naturally most people will be glad to get rid of such games for less than $5 (including shipping).

```
Biggest Loser (US,EU) (works with firmware 1.4.5) (=on all DSi's, as of 2015)
Cooking Coach (US,EU) (blocked in firmware 1.4.4 and up)
Classic Word Games (US,EU) (blocked in firmware 1.4.4 and up) (uncomfortable)
```

The cartridge contain rather small EEPROMs, barely enough to boot actual code via wifi. Wintermute's "dslink" utility is solving EEPROM size limits by storing extra boot code in the Wifi FLASH memory on DWM-W015 daughterboard (later DSi's with DWM-W024 daughterboards have small Wifi FLASH; workarounds would be to install a bigger FLASH chip in the daughterboard, or to try to squeeze the whole bootcode into the cartridge EEPROM). The different cartridges, and their localized titles are:
Biggest Loser (fitness game for fat people with dead animal food):

```
TWL-VBLV-EUU  Biggest Loser USA (UK)  ;(European title has "USA" suffix)
TWL-VBLE-USA  Biggest Loser (US)      ;(US title doesn't have "USA" suffix)
```

Cooking Coach (make healthy food with dead animals):

```
TWL-VCKE-USA  My Healthy Cooking Coach (US)
TWL-VCKV-UKV  My Cooking Coach - Prepare Healthy Recipes (UK)
TWL-VCKS-SPA  Mi Experto en Cocina - Comida Saludable (ES)
TWL-VCKF-FRA  Mon Coach Personnel - Mes Recettes Plaisir et Ligne (FR)
TWL-VCKI-ITA  Il Mio Coach di Cucina - Prepara Cibi Sani e Gustosi (IT)
TWL-VCKD-NOE  Mein Koch-Coach - Gesund und Lecker Kochen (DE)
```

Classic Word Games (crossword game) (more expensive, and less comfortable to use than Cooking Coach and Biggest Loser):

```
TWL-VCWE-USA Classic Word Games (US)
TWL-VCWV-UKV Classic Word Games (UK/EU)
```

The UK, ES, FR, IT, DE versions are working on all european consoles.

DSi Exploit Restrictions (for anything except Unlaunch exploit)
Initial memory content and register settings may be changed by the exploited game (and/or by the exploit's boot code), so the DSi won't be in the same state as when booting real licensed games.
Exploits are booted with some hardware features disabled:

```
ARM7 cannot access to SCFG/MBK configuration and Console ID registers
For DSiware Exploits: Cannot access DS Cartridge slot
For Cartridge Exploits: Cannot access SD/MMC registers
```

Currently, the only known way to get control of that hardware features would be using scanlime's Main Memory Hack (ie. soldering about 50 wires to the DSi mainboard).

Whitelist exploit (unreleased)
Rocketlauncher was supposed to get full control over the DSi via buffer overflows during NDS cart whitelist checks (via invalid Offset+Length values in the "NDHI" section of the Whitelist file, matched to the currently inserted NDS cartridge).
Unfortunately, that exploit was never finished/released - and, as by now, Unlaunch.dsi can do the same thing (even earlier after power-up, working with all retail firmware versions/regions, and without needing a NDS cart to trigger the whitelist check).

DSi Flashcarts
The majority of DSi Flashcarts are "DSi compatible", which does barely mean that they can boot DS games on DSi consoles in DS mode (but cannot run DSi software in DSi mode). The only flashcart with "DSi mode" support is/was CycloDS iEvolution (based on Cooking Coach exploit, without support for newer firmwares). Essentially, flashcarts aren't of too much use: Booting from built-in SD Card slot would also work without flashcarts, and Wifi boot would be more comfortable anyways.

DSi Regions

There are six DSi regions.

JP Region - 1 country, 1 language
Languages:
```
  Japanese (only japanese, there is no language option at all)
```
Contries:
```
  01h Japan (only japan, there is no country option at all)
```

Chinese Region (iQue)
Languages: Unknown (supposedly Chinese/Mandarin?, and maybe English or so)
Contries: Unknown (supposedly China only)
```
  A0h China?
```
China appears to refer to chinese mainland only (hongkong/taiwan are reportedly selling japanese consoles, and chinese mainland users may have imported consoles before iQue DSi launch - meaning that many chinese speaking users will be hardly able to play chinese DSi games; unless the games were released as "region-free" titles).

Korean Region - 1 country, 1 language
Languages:
```
  Koerean (only korean, there is no language option at all)
```
Contries:
```
  88h Korea (only korea, there is no country option at all)
```
Korea appears to refer to South Korea only (ie. there are no separate country/parental settings for North Korea).

Australia Region - 2 countries, 1 language (as of DSi Firmware Ver 1.4.5A)
Languages:
```
  English (only english, there is no language option at all)
```
Countries:
```
  41h Australia
  5Fh New Zealand
```

US Region - 47 countries, 3 languages (as of DSi Firmware Ver 1.3U)
Languages:
```
  English
  Francais (=French)
  Espanol  (=Spanish)
```
Countries:
```
  08h Anguilla
  09h Antigua and Barbuda
  0Ah Argentina
  0Bh Aruba
  0Dh Barbados
  0Eh Belize
  0Fh Bolivia
  10h Brazil
  11h British Virgin Islands
  12h Canada
```

```
13h Cayman Islands
14h Chile
15h Colombia
16h Costa Rica
17h Dominica
18h Dominican Republic
19h Ecuador
1Ah El Salvador
1Bh French Guiana
1Ch Grenada
1Dh Guadeloupe
1Eh Guatemala
1Fh Guyana
20h Haiti
21h Honduras
22h Jamaica
23h Martinique
24h Mexico
25h Montserrat
26h Netherlands Antilles
27h Nicaragua
28h Panama
29h Paraguay
2Ah Peru
2Bh Saint Kitts and Nevis
2Ch Saint Lucia
2Dh Saint Vincent and the Grenadines
99h Singapore
2Eh Suriname
0Ch The Bahamas
2Fh Trinidad and Tobago
30h Turks and Caicos Islands
A8h United Arab Emirates
31h United States
32h Uruguay
33h US Virgin Islands
34h Venezuela
```

Europe Region - 47 countries, 5 languages (as of DSi Firmware Ver 1.4E)

Languages:

```
English
Francais (=French)
Deutsch  (=German)
Espanol  (=Spanish)
Italiano (=Italian)
```

Countries:

```
40h Albania
42h Austria
43h Belgium
44h Bosnia and Herzegovnia
45h Botswana
46h Bulgaria
47h Croatia
48h Cyprus
49h Czech Republic
4Ah Denmark
4Bh Estonia
4Ch Finland
4Dh France
4Eh Germany
4Fh Greece
50h Hungary
51h Iceland
52h Ireland
```

```
  53h Italy
  54h Latvia
  55h Lesotho
  56h Liechtenstein
  57h Lithuania
  58h Luxembourg
  59h Macedonia
  5Ah Malta
  5Bh Montenegro
  5Ch Mozambique
  5Dh Namibia
  5Eh Netherlands
  60h Norway
  61h Poland
  62h Portugal
  63h Romania
  64h Russia
  65h Serbia
  66h Slovakia
  67h Slovenia
  68h South Africa
  69h Spain
  6Ah Swaziland
  6Bh Sweden
  6Ch Switzerland
  6Dh Turkey
  6Eh United Kindgom
  6Fh Zambia
  70h Zimbabwe
```

Note that Nintendo might expand those regions (for example, newer 'european' firmware versions might include additional african countries; unless those missing countries are already part of other/unknown regions).
The purpose of the "Country" option is unknown (maybe Nintendo has servers in different countries for online games). One known effect is that the Age Ratings in "Parental Controls" are localized per country (eg. "USK ab 18" for germany, or "18'TM" for france/finland).
The "Language" option affects the Firmware GUI, and the game title (from Icon/Title structure). Some games are also (trying to) adopt the Firmware's language setting for choosing the in-game language; but that can fail badly if the game doesn't support the selected language.

Additional Wii Regions (unsupported on DSi)

```
 Europe/Africa:
  71h 113: Azerbaijan
  72h 114: Mauritania (Islamic Republic of Mauritania)
  73h 115: Mali (Republic of Mali)
  74h 116: Niger (Republic of Niger)
  75h 117: Chad (Republic of Chad)
  76h 118: Sudan (Republic of the Sudan)
  77h 119: Eritrea (State of Eritrea)
  78h 120: Djibouti (Republic of Djibouti)
  79h 121: Somalia (Somali Republic)
 Southeast Asia:
  80h 128: Taiwan
  90h 144: Hong Kong
  91h 145: Macao
  98h 152: Indonesia
  9Ah 154: Thailand
  9Bh 155: Philippines
  9Ch 156: Malaysia
 Middle East:
  A9h 169: India
  AAh 170: Egypt
  ABh 171: Oman
  ACh 172: Qatar
  ADh 173: Kuwait
```

```
  AEh 174: Saudi Arabia
  AFh 175: Syria
  B0h 176: Bahrain
  B1h 177: Jordan
```
http://wiibrew.org/wiki/Country_Codes

3DS Reference

3DS Summary
3DS Memory and I/O Map

3DS Hardware Registers
3DS MISC Registers
3DS GPIO Registers
3DS Crypto Registers
3DS DMA Registers
3DS Config Registers
3DS SPI and I2C Bus
3DS Video
3DS Sound and Microphone
3DS Cartridge Registers
3DS Interrupts and Timers
ARM Vector Floating-point Unit (VFP)

3DS File Formats
3DS NCSD Format
3DS FIRM Format
3DS FIRM Encryption
3DS FIRM Versions
3DS FIRM Launch Parameters
3DS NCCH Format
3DS NCCH Extended Header
3DS NCCH ExeFS
3DS NCCH RomFS
3DS NCCH Encryption
3DS 3DSX Format (homebrew executables)
3DS CCAL Format (Hardware calibration, HWCAL)
3DS Title IDs
3DS Savedata Extdata
3DS Savedata Savegames
3DS Savedata DISA and DIFF
3DS Icon SMDH
3DS Banner CBMD Header
3DS Banner CGFX Graphics
3DS Banner CWAV Sound
3DS Banner Extended Banner
3DS Module NWM (Wifi Driver)
3DS Console IDs

Misc
3DS eMMC and MCU Images
3DS Component Lists
3DS Chipset Pinouts

Operating System
http://forums.nesdev.com/viewtopic.php?f=23&t=18490&p=242531#p242456
http://www.3dbrew.org/wiki/Services
http://www.3dbrew.org/wiki/Services_API

3DS Reverse Engineering
http://forums.nesdev.com/viewtopic.php?f=23&t=18490

Credits
http://www.3dbrew.org/wiki/IO_Registers

3DS Memory and I/O Map

ARM11 Memory Map

Old3DS	Address	Size		Description
Yes	00000000h	10000h		ARM11 Bootrom mirror ;also ITCM in ARM11 ?
Yes	00010000h	10000h	64K	ARM11 Bootrom
Yes	10000000h	?		IO "memory"
Yes	17E00000h	2000h	8K	MPCore private memory region (aka... IRQ ???)
No	17E10000h	1000h	4K	New3DS: L2C-310 Level 2 Cache Controller (2MB)
Yes	18000000h	600000h	6M	VRAM (two 3Mbyte banks, VRAM_A and VRAM_B)
No	1F000000h	400000h	4M	New3DS: extra memory (maybe VRAM and/or QTM?)
Yes	1FF00000h	40000h	256K	Teak DSP Code memory (aka Shared WRAM)
Yes	1FF40000h	40000h	256K	Teak DSP Data memory (aka Shared WRAM)
Yes	1FF80000h	80000h	512K	AXI WRAM
Yes	20000000h	8000000h	128M	FCRAM
No	28000000h	8000000h	128M	New3DS: FCRAM extension
Yes	FFFF0000h	10000h		ARM11 Bootrom mirror

Note: ARM11 MMU supports virtual memory addresses (so above physical addresses may appear elsewhere in virtual memory).

ARM9 Memory Map

Old3DS	Address	Size		Description
Yes	00000000h	"8000000h"		Instruction TCM, mirrored each 8000h bytes
Yes	01FF8000h	8000h	32K	Instruction TCM (used here by kernel & titles)
Yes	07FF8000h	8000h		Instruction TCM (used here by bootrom)
Yes	08000000h	100000h	1M	ARM9-only internal memory (and ARM7 regions)
No	08100000h	80000h	512K	New3DS:ARM9-only extension (if any/if enabled)
Yes	10000000h	8000000h		IO "memory"
Yes	18000000h	600000h	6M	VRAM (two 3Mbyte banks, VRAM_A and VRAM_B)
Yes	1FF00000h	40000h	256K	Teak DSP Code memory (aka Shared WRAM)
Yes	1FF40000h	40000h	256K	Teak DSP Data memory (aka Shared WRAM)
Yes	1FF80000h	80000h	512K	AXI WRAM
Yes	20000000h	8000000h	128M	FCRAM
No	28000000h	8000000h	128M	New3DS: FCRAM extension
Yes	FFF00000h	4000h	16K	Data TCM (mapped here during bootrom)
Yes	FFFF0000h	10000h	64K	ARM9 Bootrom

I/O Maps

ARM9-only Registers

Physaddr	Old3DS	A9/A11	Category	
10000000h	Yes	A9	CONFIG9 Registers	
10001000h	Yes	A9	IRQ Registers	
10002000h	Yes	A9	NDMA Registers	DMA (alike DSi's NDMA)
10003000h	Yes	A9	TIMER Registers	Timers (alike GBA/NDS/DSi)
10004000h	Yes	A9	CTRCARD Registers	ROM cart in 3DS mode
10005000h		A9	CTRCARD?	2nd ROM cart slot?

```
10006000h  Yes     A9      SDMMC Registers    For eMMC and SD Card slot
10007000h          A9      SDxx?              ?
10008000h  Yes     A9      PXI Registers      aka IPC
10009000h  Yes     A9      AES Registers      Crypto
1000A000h  Yes     A9      SHA Registers      Crypto
1000B000h  Yes     A9      RSA Registers      Crypto
1000C000h  Yes     A9      XDMA Registers     DMA
1000D000h  Yes     A9      SPI_CARD Registers Savedata in ROM carts
10010000h  Yes     A9      CONFIG Registers   More CONFIG9 registers
10011000h  Yes     A9      PRNG Registers     Pseudo Random Generator
10012000h  Yes     A9      OTP Registers      Console IDs
10018000h  Yes     A9      ARM7 Registers     GBA/NDS/DSi mode config
```

ARM11/ARM9 Registers

Below registers can be accessed by both ARM11 and ARM9. However, most of them are working best on ARM11 side (because IRQ and DMA startup modes are implemented on ARM11 side only; and, in some cases, even FIFOs are in a separate ARM11-only area at 10300000h and up).
One exception is NTRCARD (this does have IRQ/DMA on both ARM9 and ARM11 side).

```
10100000h  Yes     A11/A9  Debug WIFI SDIO Regs?  ;uh, actually zerofilled
10101000h  Yes     A11/A9  HASH Registers     Crypto (same as SHA)
10102000h  Yes     A11/A9  Y2R_0 Registers    First YUV-to-RGBA
10103000h  Yes     A11/A9  CSND Registers     Sound channels and capture
10110000h  Yes     A11/A9  LGYFB_0            Legacy GBA/NDS Video
10111000h  Yes     A11/A9  LGYFB_1            Legacy GBA/NDS Video
10120000h  Yes     A11/A9  Camera Registers   Camera Bus 0 (DSi cameras)
10121000h  Yes     A11/A9  Camera Registers   Camera Bus 1 (left-eye)
10122000h  Yes     A11/A9  WIFI Registers     SDIO Wifi
10123000h  Yes     A11/A9  ?                  SDIO?   ;uh, actually data.abt
10130000h  No      A11/A9  L2B_0              New3DS: First RGB-to-RGBA
10131000h  No      A11/A9  L2B_1              New3DS: Second RGB-to-RGBA
10132000h  No      A11/A9  Y2R_1              New3DS: Second YUV-to-RGBA
10140000h  Yes     A11/A9  CONFIG11 Registers
10141000h  Yes     A11/A9  CONFIG11 Registers
10142000h  Yes     A11/A9  SPI Registers      SPI Bus1 (Tsc)
10143000h  Yes     A11/A9  SPI Registers      SPI Bus2 (unused)
10144000h  Yes     A11/A9  I2C Registers      I2C Bus1 (for 3DS devices)
10145000h  Yes     A11/A9  CODEC Registers    ?
10146000h  Yes     A11/A9  HID Registers      Keypad
10147000h  Yes     A11/A9  GPIO Registers
10148000h  Yes     A11/A9  I2C Registers      I2C Bus2 (for 3DS gimmicks)
10160000h  Yes     A11/A9  SPI Registers      SPI Bus0 (Pwrman,WifiFlash,Tsc)
10161000h  Yes     A11/A9  I2C Registers      I2C Bus0 (for DSi devices)
10162000h  Yes     A11/A9  MIC Registers      Microphone
10163000h  Yes     A11/A9  PXI Registers      aka IPC
10164000h  Yes     A9/A11  NTRCARD Registers  ROM Cart in NDS/DSi mode
10165000h  Yes     A11/A9  MP Registers       NDS-Wifi WIFIWAITCNT(10165206h)
10170000h  Yes     A11/A9  MP Registers       NDS-Wifi WS0 Area (8000h bytes)
10178000h  Yes     A11/A9  MP Registers       NDS-Wifi WS1 Area (8000h bytes)
10180000h                                     (end of above area)
```

ARM11-only Registers

```
10200000h  Yes     A11     CDMA               DMA
10201000h  Yes     A11     UNKNOWN            ?
10202000h  Yes     A11     LCD Registers      LCD
10203000h  Yes     A11     DSP Registers      Teak DSP (if enabled)
10204000h  Yes     A11     UNKNOWN            ?
10206000h  No      A11     CDMA               New3DS: DMA
10207000h  No      A11     MVD Registers      New3DS: Movie Decoder or so?
1020F000h  Yes     A11     AXI                ?
10300000h  Yes     A11     FIFO? debug?  Maybe debug wifi FIFOs or so?
10301000h  Yes     A11     FIFO HASH (SHA)          contains FIFO
10302000h  Yes     A11     FIFO Y2R_0
10310000h  Yes     A11     FIFO LGYFB_0
```

```
10311000h  Yes     A11     FIFO LGYFB_1
10320000h  Yes     A11     FIFO Camera Bus 0 (20h-word window FIFO)
10321000h  Yes     A11     FIFO Camera Bus 1 (20h-word window FIFO)
10322000h  Yes     A11     FIFO? wifi??  Maybe contains DATA32 FIFOs or so?
10323000h  ?       ?       data abort (maybe SDIO FIFO for 10123000h, if any)
10330000h  No      A11     FIFO L2B_0     New3DS: L2B_0 FIFO (empty=data_abt)
10331000h  No      A11     FIFO L2B_1     New3DS: L2B_1 FIFO (empty=data_abt)
10332000h  No      A11     FIFO MVD Y2R_1 New3DS: MVD Y2R FIFO
10400000h  Yes     A11     GPU External Registers
10401000h  Yes     A11     GPU Internal Registers
17E00000h 100h     A11     MPCore SCU (Snoop Control Unit)
17E00100h 100h     A11     MPCore CPU Interrupt Interface
17E00200h 100h     A11     MPCore CPU0 Interrupt Interface ;\
17E00300h 100h     A11     MPCore CPU1 Interrupt Interface ; (aliased for
17E00400h 100h     A11     MPCore CPU2 Interrupt Interface ; debug purposes)
17E00500h 100h     A11     MPCore CPU3 Interrupt Interface ;/
17E00600h 100h     A11     MPCore CPU Timer and Watchdog
17E00700h 100h     A11     MPCore CPU0 Timer and Watchdog  ;\
17E00800h 100h     A11     MPCore CPU1 Timer and Watchdog  ; (aliased, too?)
17E00900h 100h     A11     MPCore CPU2 Timer and Watchdog  ;  <-- N/A in 3DS?
17E00A00h 100h     A11     MPCore CPU3 Timer and Watchdog  ;/ <-- N/A in 3DS?
17E00B00h 500h     A11     MPCore Reserved (access causes a DECERR abort)
17E01000h 1000h    A11     MPCore Global Interrupt Distributor
17E10000h  No      A11     New3DS: L2C-310 Level 2 Cache Controller
```

IO registers starting at physical address 10200000h are not accessible from the ARM9 (which includes all LCD/GPU registers). It seems IO registers below physical address 10100000h are not accessible from the ARM11 bus (=trigger data abort, as so on all unused 1000h-byte areas).

ARM11 kernel virtual address mappings for these registers varies for different builds. For ARM11 user mode applications you have:

```
physaddr = virtaddr-1EC00000h+10100000h
```

That is:

```
Physical   Virtual
101xxxxxh  1ECxxxxxh
102xxxxxh  1EDxxxxxh
```

3DS MISC Registers

```
_____________________________ UNKNOWN Registers _____________________________
```

Unknown Registers at 10201000h

```
10201000h 4    Unknown R/W=0000001fh (res=00000001h)
10201004h 0Ch  Unused (0)
10201010h 4    Unknown R/W=00070003h (res=00020000h)
10201014h 0Ch  Unused (0)
10201020h 4    Unknown R/W=80000f0fh (res=00000808h)  ;XXX or zero?
10201024h FDCh Unused (0)
```

Unknown Registers at 10204000h

```
10204000h 4    Unknown R/W=000000F1h (res=00000000h)
10204004h 4    Unknown, readonly (00000001h) (R)
10204008h 08h  Unused (0) ?
10204010h 4    Unknown, readonly (00000202h) (R)
10204014h 4    Unknown, readonly (12000802h) (R) (or data abort!)
10204018h 08h  Unused (0) ?
10204020h 4    Unknown, readonly (00000020h) (R) ;or res=00000000h?
10204024h FDCh Unused (0) ?
```

FCRAM Configuration Registers

The 3DS uses the following FCRAM chips:

```
  Old3DS: Fujitsu MB82M8080-07L (128Mbyte)
  New3DS: Fujitsu 82MK9A9A, 7L (256Mbyte)
```
The chip in Old3DS does reportedly internally contain 2 dies:
```
  MB81EDS516545 (4x2Mx64bit, aka 64Mbyte)
  MB82DBS08645 (unknown, supposedly contains the missing 64Mbyte)
```
The MB81EDS516545 datasheet says that there are some internal config registers, which must be initialized after power up, the initialization requires changing /RAS and /CAS and some other pins. Unknown if/how the 3DS hardware can initialize those registers.
Using FCRAM in GBA Mode (for emulating the GBA ROM-image) is highly unstable, unknown how to configure more stable behaviour.

_____ SD/MMC Registers _____

SD/MMC Registers
```
  10006000h A9      SD/MMC (for SD/MMC Slot and internal eMMC)
  10006200h A9      mirrors of above (each 200h bytes)
  10007000h A9      SDxx (unused? or maybe SDIO wifi on ARM9 side?)
  10007200h A9      mirrors of above (each 200h bytes)
  10100000h A11/A9  Debug SDIO Regs? (actually just zerofilled?)
  10122000h A11/A9  SDIO WIFI? this is wifi?
  10123000h A11/A9  Reportedly SDIO? (actually just data abort?)
```
These registers are used to access the system NAND and the inserted SD card. Both devices use the same interface.
HCLK of the SDMMC controller is 67.027964 MHz (double of the DSi HCLK).
Unknown if eMMC and Wifi are fast enough to support HCLK/2 on 3DS.
Standard SD cards may require HCLK/4 on 3DS (unless the card was detected to support HCLK/2; don't know if it requires further card initialization to enable fast mode?).

IRQ_STAT
```
  3     SD card removal flag     (Set to 1 when SD card is removed)
  4     SD card insertion flag   (Set to 1 when SD card is inserted)
  5     SD card insertion status (0=Missing, 1=Inserted)
```
SD card insertion status
Assertion happens around 250 ms after SD card insertion and/or enabling the EMMC hardware (delay could possibly be due to an SD bus timeout?).

Wifi Registers
These registers are used to control the Wifi card via the SDIO protocol.
```
  Address  Width  Name
  10122000h 2  WIFI_CMD
  10006002h 2  uh, at 10006002h?
  10122004h 2  WIFI_CMDARG0
  10122006h 2  WIFI_CMDARG1
  1012200ah 2  WIFI_BLKCOUNT
  10122024h 2  WIFI_CLKCTL
  10122026h 2  WIFI_BLKLEN
```
XXX DATA32 FIFO might be at 10322000h (instead of 1012210Ch)... ?

_____ IPC Registers _____

PXI Registers
```
  Address Width Old3DS Name         Used by
  10008000h 4   Yes   PXI_SYNC9     Boot9, Process9     ;-SYNC
  10008004h 2   Yes   PXI_CNT9      Boot9, Process9     ;\
  10008008h 4   Yes   PXI_SEND9                         ; FIFO
  1000800Ch 4   Yes   PXI_RECV9                         ;/
  10163000h 4   Yes   PXI_SYNC11    Boot11              ;-SYNC
  10163004h 2   Yes   PXI_CNT11     Boot11              ;\
```

```
  10163008h 4   Yes   PXI_SEND11                        ; FIFO
  1016300Ch 4   Yes   PXI_RECV11                        ;/
```
The PXI registers are similar to those on DS. Uh, aka what is known as IPC.
But, except, SYNC is 8bit (instead 4bit)!
The FIFO registers seem to be exactly same as on NDS.

10008000h - ARM9 - PXI_SYNC9
10163000h - ARM11 - PXI_SYNC11
```
  0-7   R    Data received from remote SYNC bit8-15
  8-15  W    Data sent to remote SYNC bit0-7 (CAUTION: write-only, unlike NDS!)
  16-22 -    Unused (0)
  23    -    Unused (0)   ;<-- reportedly "?" whatever that means, if anything?
  24-28 -    Unused (0)
  29    -    PXI_SYNC11: Unused (0)
  30    W    PXI_SYNC11: Send IRQ to ARM9 IF.bit12 (0=No change, 1=Yes)
  29    W    PXI_SYNC9: Send IRQ to ARM11 IRQ 50h  (0=No change, 1=Yes)
  30    W    PXI_SYNC9: Send IRQ to ARM11 IRQ 51h  (0=No change, 1=Yes)
  31    R/W  Enable IRQ from remote CPU            (0=Disable, 1=Enable)
```
Caution: The send-value (in bit8-15) is write-only on 3DS (reads as zero), this is different as on NDS/DSi. As workaround: Use 8bit LDRB/STRB instead of 32bit LDR/STR when changing the IRQ bits.

10008004h - ARM9 - PXI_CNT
10163004h - ARM11 - PXI_CNT
```
  0     R    Send Fifo Empty Status      (0=Not Empty, 1=Empty)
  1     R    Send Fifo Full Status       (0=Not Full, 1=Full)
  2     R/W  Send Fifo Empty IRQ         (0=Disable, 1=Enable)
  3     W    Send Fifo Clear             (0=Nothing, 1=Flush Send Fifo)
  4-7        unknown/unspecified
  8     R    Receive Fifo Empty          (0=Not Empty, 1=Empty)
  9     R    Receive Fifo Full           (0=Not Full, 1=Full)
  10    R/W  Receive Fifo Not Empty IRQ  (0=Disable, 1=Enable)
  11-13      unknown/unspecified
  14    R/W  Error, Read Empty/Send Full (0=No Error, 1=Error/Acknowledge)
  15    R/W  Enable Send/Receive Fifo    (0=Disable, 1=Enable)
```

10008008h - ARM9 - PXI_SEND
10163008h - ARM11 - PXI_SEND
1000800Ch - ARM9 - PXI_RECV
1016300Ch - ARM11 - PXI_RECV
FIFO
DS Inter Process Communication (IPC)

ARM11-to-ARM11 Messages
ARM11 cores can use the Software Interrupt Register, Port 17E01F00h (and data in AXI memory or other RAM locations) to communicate with each other.

_____ HID Registers _____

HID Registers (aka Keypad)
```
  10146000h 2  HID_PAD
  10146002h 2  HID_PAD_IRQ
```

10146000h - HID_PAD (R)
Same as NDS/DSi, but with X/Y buttons included right in this register.
```
  0     Button A        (0=Pressed, 1=Released)
  1     Button B
  2     Select
  3     Start
  4     DPAD Right
```

```
  5     DPAD Left
  6     DPAD Up
  7     DPAD Down
  8     Button R
  9     Button L
  10    Button X
  11    Button Y
  12-15 Unused (0)
```

10146002h - HID_PAD_IRQ (R/W)

```
  0-11  IRQ Buttons     (0=Ignore, 1=Select)
  12-13 Unused (0)
  14    IRQ Enable Flag (0=Disable, 1=Enable, Interrupt 5Bh)
  15    IRQ Condition   (0=Logical OR, 1=Logical AND)
```

Similar as Port 4000132h on GBA/NDS, used as wake-up source in sleep mode.

Other Inputs

```
  Circle Pad: Connected to Touchscreen/Sound controller
  Power/Home buttons: see I2C MCU
  Wifi switch/slider/button: see I2C MCU (older 3DS only?)
  Volume/3D Sliders: see I2C MCU
  Accelerometer: see I2C MCU (also includes pedometer step counter)
  Gyroscope: see I2C Gyroscope
  Hinge/Shell: see GPIO (and/or reportedly also I2C MCU)
  DebugPad: See I2C (whatever that is)
  MIC, Cameras, IR, QTM, Charge, SD slot, ROM slot
  Old3DS: NFC reader/writer (external adaptor, instead of New3DS built-in)
  Old3DS: Circle Pad Pro (2nd Circle Pad and R/ZL/ZR buttons, on IR port)
  New3DS: C-stick and ZL/ZR: see I2C (instead of Old3DS's circle pad pro)
  New3DS: NFC built-in (instead of Old3DS's external adaptor)
  New3DS: Head-tracking (whatever that is, probably just camera with IR-LED..?)
  New3DS: Invisible camera IR-LED (unknown how to control/test that)
```

_____ MP? Registers _____

MP Registers (local wifi multiplayer?)

```
  10165206h 2      NDS-Wifi WIFIWAITCNT
  10170000h 8000h  NDS-Wifi WS0 region
  10178000h 8000h  NDS-Wifi WS1 region
```

These registers are used by MP-module. MP-module seems to be used for 3DS<>DSi local-WLAN communications?

_____ PARODY Registers _____

Parody Register summary

After reading through many official datasheets and homebrew wiki pages, some of the more amazing 3DS hardware related information goes into 1020B000h...

```
  1020B000h 4   Certain Register
  1020B014h 4   Assert Register
  1020B034h 4   This Register
  1020B096h 2   Figure Register
  1020B1BFh 1   Byte Register
  1020B2E4h ??  Broken Register
  1020B3D0h 4   Snakish Register
```

1020B000h - Certain Register

This is the Certain register. The register contains certain bits that are used to enable certain functions, each bit must be set to a certain value. The entire system will hang if used with a certain setting.

1020B014h - Assert Register

Bits in the Assert register get asserted when asserting events are asserted and de-asserted. You are asserted to assert what the asserted information does assert. Assert your dictionary if you can't assert that.

1020B034h - This Register
This register can be used to change the value of this register by writing to this register.

1020B096h - Figure Register
Figure 128 contains information about the Figure register. These register specifications are outlined in Figure 128. See Figure 128 for details about the Figure Register in Figure 128.
Figure 128: Figure 128 is left blank intentionally.

1020B0DCh - ?

1020B1BFh - Byte Register
8-bit Byte value 0,1: 1-ish enable 2:31 must be <this&0x3<=2'b010 +! Pdn SoC value>.

1020B2E4h - Broken Register
This works exactly as described in the broken link on the hardware page.

1020B3D0h - Snakish Register
With Native_firm as of 8.1.0.192 three additional unsigned flags have been introduced to the old(6.5.0.0+) specification which did have certain bits at a specific virtual address and that were not formerly asserted in 1.0.2.30 with *ANY* firm these values must be acquired with strict binary point comparisions(in the same format as the other bits) prohibitvely permitting abstractions of the reorganized kernel architecture in the *actual* snakish sentences used to describe the entire register space(that CAN be changed in future).

3DS GPIO Registers

_____ GPIO Registers _____

GPIO Registers

```
Address  Width Name
10147000h 2   GPIO_DATA0
10147010h 4x1 GPIO_DATA1
10147014h 2   GPIO_DATA2
10147020h 4x2 GPIO_DATA3
10147028h 2   GPIO_DATA4_WIFI
10147100h ..  Legacy RTC ?
```

10147000h - GPIO_DATA0_DATA_IN (R?)

```
0      Unknown (0=?, 1=Normal) HID-sysmodule, HID PAD state  ;\GPIO services
1      Touchscreen (0=Pen Down, 1=No)        (IRQ:63h)       ; bitmask 7h
2      Hinge (0=Shell Open, 1=Shell Closed)  (IRQ:60h/62h?)  ;/
3      Unused?
4      Unused?  ;"Only used by Boot11" (uh, but bootrom tests only bit2)
5-15   Unused?
```

10147010h - GPIO_DATA1_DATA_IN (R)
10147010h - GPIO_DATA1_DATA_OUT (W)
10147011h - GPIO_DATA1_DIRECTION (R/W)
10147012h - GPIO_DATA1_IRQ_EDGE (R/W) ?
10147013h - GPIO_DATA1_IRQ_ENABLE (R/W)

```
0      Headphone (0=None, 1=Connected) gpio:CDC (IRQ:64h) ;\GPIO services
1      Unknown   (0=?, 1=Normal)               (IRQ:66h) ;/bitmask 18h
2-7    Unused
```

10147014h - GPIO_DATA2_DATA_OUT (R/W)

```
0      gpio:MCU, gpio:NWM: Wifi?     ;-GPIO services bitmask 20h
1-15   Unused
```

10147020h - GPIO_DATA3_DATA_IN (0=Low, 1=High) (R)
10147020h - GPIO_DATA3_DATA_OUT (0=Low, 1=High) (W)
10147022h - GPIO_DATA3_DIRECTION (0=Normal/Input, 1=Output) (R/W)
10147024h - GPIO_DATA3_IRQ_EDGE (0=Falling, 1=Rising) (R/W) ?
10147026h - GPIO_DATA3_IRQ_ENABLE (0=Disable, 1=Enable) (R/W)

Most of these might be IRQ signals from I2C peripherals?
Stuff like QTM, NFC, ir:rst is New3DS only, so Old3DS is probably different here?

```
0      C-stick! gpio:CDC, gpio:IR <-- 1 after ir:rst read(IRQ:68h?);\
1      IrDA IRQ gpio:IR   (0=IRQ)   Boot11(uh, really?)  (IRQ:69h) ;
2      Gyro IRQ gpio:HID  (1=IRQ, in default cfg)        (IRQ:6Ah?); GPIO
3      ?        gpio:HID, gpio:IR   used with ir:rst.    (IRQ:6Bh) ; services
4      ?        gpio:IR send? 1=IR LED enable, 0=disable (IRQ:6Ch) ; bitmask
5      ?        gpio:IR receive?                         (IRQ:6Dh) ; 3FFC0h
6      ?        gpio:NFC                                 (IRQ:6Eh) ;
7      ?        gpio:NFC                                 (IRQ:6Fh) ;
8      ?        gpio:HID  HID-sysmodule, HID PAD state   (IRQ:70h?);
9      MCU IRQ  gpio:MCU      (MCU irq, interrupt 71h)   (IRQ:71h?);
10     NFC?     gpio:NFC  <-- cleared after NFC read     (IRQ:72h) ;
11     ?        gpio:QTM   (Twlbg? and/or New3DS?)       (IRQ:73h?);/
12-15  Unused (0)
```

10147028h - GPIO_DATA4_DATA_OUT_WIFI (R/W)

```
0      Wifi Enable (1=On) ;gpio:MCU, gpio:NWM ;-GPIO services bitmask 40000h
1-15   Unused (0)
```

Default GPIO values

After bootrom initialization, these are the values of the registers:

```
Address    Value
10147000h  0003h
10147010h  00000002h
10147014h  0000h
10147020h  00000DFBh
10147024h  00000000h
10147028h  0000h
```

10147100h - Legacy RTC ?

Normally only 10147100h.bit15 is R/W, and the other bits/registers at 101471xxh are all zero. But reportedly they can contain this...

```
 10147100h 2      RTC_CNT          Control register
Below are all bytewise bitswapped...
 10147110h 1      RTC_REG_STAT1    Rtc status register 1                0
 10147111h 1      RTC_REG_STAT2    Rtc status register 2                1
 10147112h 1      RTC_REG_CLKADJ   Rtc clock adjustment register         6
 10147113h 1      RTC_REG_FREE     The free general purpose rtc register  7
 10147120h 4      RTC_REG_TIME1    Seconds, minutes, hours, day of week
 10147124h 4 (3?) RTC_REG_TIME2    Day, month and year
 10147130h 4 (3?) RTC_REG_ALRMTIM1 Rtc alarm time register 1             4
 10147134h 4 (3?) RTC_REG_ALRMTIM2 Rtc alarm time register 2             5
Below are for DSi, unknown if they are used/enabled for NDS, too...
 10147140h 4 (3?) RTC_REG_COUNT    Rtc DSi counter register             ex0
 10147150h 1      RTC_REG_FOUT1    Rtc DSi fout register 1              ex1
 10147151h 1      RTC_REG_FOUT2    Rtc DSi fout register 2              ex2
 10147160h 4 (3?) RTC_REG_ALRMDAT1 Rtc DSi alarm date register 1        ex4
 10147164h 4 (3?) RTC_REG_ALRMDAT2 Rtc DSi alarm date register 2        ex5
In single bytes...
 10147110h 1      RTC_REG_STAT1         Rtc status register 1
```

```
10147111h 1      RTC_REG_STAT2          Rtc status register 2
10147112h 1      RTC_REG_CLKADJ         Rtc clock adjustment register
10147113h 1      RTC_REG_FREE           Free general purpose register
10147114h 0Ch    Unused
10147120h 1      RTC_REG_TIME_SECOND    Second
10147121h 1      RTC_REG_TIME_MINUTE    Minute
10147122h 1      RTC_REG_TIME_HOUR      Hour
10147123h 1      RTC_REG_TIME_DOW       Day of week
10147124h 1      RTC_REG_TIME_DAY       Day
10147125h 1      RTC_REG_TIME_MONTH     Month
10147126h 1      RTC_REG_TIME_YEAR      Year
10147127h 09h    Unused
10147130h 4 (3?) RTC_REG_ALARM1_MINUTE  Alarm 1 Minute
10147131h        RTC_REG_ALARM1_HOUR    Alarm 1 Hour
10147132h        RTC_REG_ALARM1_DOW     Alarm 1 Day of week
10147133h 01h    Unused
10147134h 4 (3?) RTC_REG_ALARM2_MINUTE  Alarm 2 Minute
10147135h        RTC_REG_ALARM2_HOUR    Alarm 2 Hour
10147136h        RTC_REG_ALARM2_DOW     Alarm 2 Day of week
10147137h 09h    Unused
10147140h 4 (3?) RTC_REG_COUNT_LSB      Rtc DSi Counter LSB?
10147141h        RTC_REG_COUNT_MID      Rtc DSi Counter MID
10147142h        RTC_REG_COUNT_MSB      Rtc DSi Counter MSB?
10147143h 0Dh    Unused
10147150h 1      RTC_REG_FOUT1          Rtc DSi Fout register 1
10147151h 1      RTC_REG_FOUT2          Rtc DSi Fout register 2
10147152h 0Eh    Unused
10147160h 4 (3?) RTC_REG_ALARM1_DAY     Rtc DSi Alarm 1 Day
10147161h        RTC_REG_ALARM1_MONTH   Rtc DSi Alarm 1 Month
10147162h        RTC_REG_ALARM1_YEAR    Rtc DSi Alarm 1 Year
10147163h 01h    Unused
10147164h 4 (3?) RTC_REG_ALARM2_DAY     Rtc DSi Alarm 2 Day
10147165h        RTC_REG_ALARM2_MONTH   Rtc DSi Alarm 2 Month
10147166h        RTC_REG_ALARM2_YEAR    Rtc DSi Alarm 2 Year
10147167h ..     Unused
```

That is probably for NDS/DSi RTC (4000138h). For GBA Cart RTC see "ARM7_RTC_xxx" registers at 100181xxh.

10147100h - RTC_CNT

```
0     Latch STAT1                                                  (W)
1     Latch STAT2                                                  (W)
2     Latch CLKADJ                                                 (W)
3     Latch FREE                                                   (W)
4     Latch TIME                                                   (W)
5     Latch ALRMTIM1                                               (W)
6     Latch ALRMTIM2                                               (W)
7     Latch COUNT                                                  (W)
8     Latch FOUT1                                                  (W)
9     Latch FOUT2                                                  (W)
10    Latch ALRMDAT1                                               (W)
11    Latch ALRMDAT2                                               (W)
12    ARM7 Busy? This may be chipselect                            (R)
13    ARM7 write command received? (writing 1 clears it seems)     (R/ack)
14    ARM7 read command received? (writing 1 clears it seems)      (R/ack)
15    DS SIO SI pin (rtc irq pin)                                  (R/W)
```

Unknown if "Latch" means writing or reading the registers, or if both is somehow supported.
Unknown why RTC IRQ pin is write-able, maybe ARM11 must manually trigger it? But if the ALARM registers are write-only then ARM11 couldn't know their current value (except, by assuming that the written value wasn't changed from NDS side).

3DS Crypto Registers

3DS Crypto - AES Registers

AES Registers (ARM9)

```
  Address           Name          Byte order  Word order
  10009000h 4   R/W AES_CNT       Little      -          ;-bit22-29 unlike DSi
  10009004h 4   W   AES_BLKCNT    Little      -
  10009008h 4   W   AES_WRFIFO    WordWrite   FifoWrite
  1000900Ch 4   R   AES_RDFIFO    WordRead    FifoRead
  10009010h 1   R/W AES_KEYSEL    -           -          ;\new, unlike DSi
  10009011h 1   R/W AES_KEYCNT    -           -          ;/
  10009020h 16  W   AES_IV        WordWrite   Little
  10009030h 16  W   AES_MAC       WordWrite   Little
  10009040h 48  W   AES_KEY0      WordWrite   Little
  10009070h 48  W   AES_KEY1      WordWrite   Little
  100090A0h 48  W   AES_KEY2      WordWrite   Little
  100090D0h 48  W   AES_KEY3      WordWrite   Little
  10009100h 4   W   AES_KEYFIFO   WordWrite   FifoWrite  ;\
  10009104h 4   W   AES_KEYXFIFO  WordWrite   FifoWrite  ; new, unlike DSi
  10009108h 4   W   AES_KEYYFIFO  WordWrite   FifoWrite  ;/
```

Most of these registers are same as on DSi:
[DSi AES I/O Ports]
New features are more keyslots, CBC/ECB modes, optional big-endian mode, and a new Key X/Y scrambler.

10009000h - ARM9 - AES_CNT (R/W)

```
  0-4   Write FIFO Count    (00h..10h words) (00h=Empty, 10h=Full)          (R)
  5-9   Read FIFO Count     (00h..10h words) (00h=Empty, 10h=Full)          (R)
  10    Write FIFO Flush    (0=No change, 1=Flush)                  (N/A or W)
  11    Read FIFO Flush     (0=No change, 1=Flush)                  (N/A or W)
  12-13 Write FIFO DMA Size (0..3 = 16,12,8,4 words) (2=Normal=8)   (R or R/W)
  14-15 Read FIFO DMA Size  (0..3 = 4,8,12,16 words) (1=Normal=8)   (R or R/W)
  16-18 CCM MAC Size, max(4,(N*2+2)) bytes, usually 7=16 bytes      (R or R/W)
  19    CCM Pass Associated Data to RDFIFO (0=No/Normal, 1=Yes)     (R or R/W)
  20    CCM MAC Verify Source (0=From AES_WRFIFO, 1=From AES_MAC)   (R or R/W)
  21    CCM MAC Verify Result (0=Invalid/Busy, 1=Verified/Okay)          (R)
 Below bits (bit22-29) are other than DSi
  22    Byte order for Word Read  (0=Little endian, 1=Big endian)  (R? or R/W)
  23    Byte order for Word Write (0=Little endian, 1=Big endian)  (R? or R/W)
  24    Word order per Fifo Read  (0=Little first, 1=Big first)    (R? or R/W)
  25    Word order per Fifo Write (0=Little first, 1=Big first)    (R? or R/W)
  26    Key Select        (0=No change, 1=Apply key selected in AES_KEYSEL) (W)
  27-29 Mode (0=CCM decrypt, 1=CCM encrypt, 2=CTR, 3=CTR,
              4=CBC decrypt, 5=CBC encrypt, 6=ECB decrypt, 7=ECB encrypt)
  30    Interrupt Enable  (0=Disable, 1=Enable IRQ on Transfer End)  (R or R/W)
  31    Start/Enable      (0=Disable/Ready, 1=Enable/Busy)               (R/W)
```

10009010h - ARM9 - AES_KEYSEL (R/W)

```
  0-5   Keyslot for encrypt/decrypt, apply via AES_CNT.bit26 (00h..3Fh)   (R/W)
  6-7   Unused (0)
```

10009011h - ARM9 - AES_KEYCNT (R/W)

```
  0-5   Keyslot for writes via AES_KEYxyFIFO (04h..3Fh, or 0..3=None)     (R/W)
```

```
  6     Key X/Y Scrambler for key 4-3Fh      (0=3DS, 1=DSi)              (R/W)
  7     Flush AES_KEYxyFIFO                  (0=No change, 1=Flush)        (W)
```
"The hardware key generator is triggered by writing the keyY, which is the only way to trigger it with the 3DS keyslots. The algorithm for generating the normal-key from keyX and keyY is as follows":
```
  KeyDSi = (((KeyX)       XOR KeyY) + FFFEFB4E295902582A680F5F1A4F3E79h) ROL 42
  Key3DS = (((KeyX ROL 2) XOR KeyY) + 1FF9E9AAC5FE0408024591DC5D52768Ah) ROL 87
```
Key 0-3 are writeable only via the old DSi-style AES_KEY0..3 registers (not via AES_KEYxyFIFO), and key0-3 are always using the old DSi-style Key scrambler.

10009100h - ARM9 - AES_KEYFIFO - Normal Key (four words)
10009104h - ARM9 - AES_KEYXFIFO - Key X (four words)
10009108h - ARM9 - AES_KEYYFIFO - Key Y (four words)
Whatever, new (unlike DSi).
The Key FIFO's are automatically flushed after each 4 words, and after setting AES_KEYCNT.bit7, and reportedly also when changing the AES_CNT word order.

_____ Notes and DSi Registers _____

Notes
When AES_CNT.bit31 is set, then AES_CNT essentially becomes locked and doesn't change when written to. However if AES_CNT.bit26 is "set", keyslot-selection is cued to be handled when AES_CNT.bit31 is cleared. Clearing AES_CNT.bit31 while the AES engine is doing crypto will result in the AES engine stopping crypto, once it finishes processing the current block.

10009004h - ARM9 - AES_MACEXTRABLKCNT
(CCM-MAC extra data length)>>4, ie. the number of block of CCM-MAC extra data.

10009006h - ARM9 - AES_BLKCNT
(Data length)>>4, ie. the number of blocks to process

10009008h - ARM9 - AES_WRFIFO
1000900Ch - ARM9 - AES_RDFIFO
Reading from AES_RDFIFO when there's no data available in the RDFIFO will result in reading the last word that was in the RDFIFO.
When triggering either RDFIFO or WRFIFO to be flushed, the AES Engine does not clear either buffer.
Word order and endianness can be changed between each read/write to these FIFOs. However changing the word order when writing to WRFIFO can cause the word to be written outside the current block, leaving uninitialized data in its place. Attempts to change endianness or word order are not honored when reading from RDFIFO when no more data is available.

10009020h - ARM9 - AES_IV (16 bytes)
This register specifies the counter (CTR mode), nonce (CCM mode) or the initialization vector (CBC mode) depending on the mode of operation. For CBC and CTR mode this register takes up the full 16 bytes, but for CCM mode the nonce is only the first 12 bytes. The AES engine will automatically increment the counter up to the maximum BLKCNT, after which point it must be manually incremented and set again.

10009030h - ARM9 - AES_MAC (16 bytes)
This register specifies the message authentication code (MAC) for use in CCM mode.

10009040h/10009070h/100090A0h/100090D0h - AES_KEY0/1/2/3
These registers are the same as they were on TWL, and are likely preserved for compatibility reasons. The keyslot is updated immediately after *any* data(u8/u32/...) is written here, which was used on DSi to break the key-generator.

CCM mode pitfall

Non-standard AES-CCM behaviour is observed on Wrap/Unwrap function. According to RFC 3610, the first block B_0 for authentication should be generated from the message length and some other parameters. Using these function, it seems that the message length is aligned up to 16 when generating B_0. This makes the generated MAC not compliant with the standard when (inputsize-noncesize)%16!=0. It is very likely that this non-standard behaviour happens on the hardware level, but not confirmed yet.

```
uh, or does it just OMIT that, as on DSi?
instead of that "aligned" thing?
```

3DS Crypto - SHA Registers

SHA Registers

```
1000A000h 4     ARM9        SHA_CNT          ;\
1000A004h 4     ARM9        SHA_BLKCNT       ; for ARM9
1000A040h 20h   ARM9        SHA_HASH         ;
1000A080h 40h   ARM9        SHA_FIFO         ;/
10101000h 4     ARM11/ARM9  SHA_CNT          ;\for ARM11 (some registers
10101004h 4     ARM11/ARM9  SHA_BLKCNT       ; can be also accessed by ARM9,
10101040h 20h   ARM11/ARM9  SHA_HASH         ; but FIFO and DMA DRQs are
10301000h 40h   ARM11       SHA_FIFO         ;/working for ARM11 only)
```

1000A000h/10101000h - SHA_CNT - SHA Control (R/W)

```
0     Read: IN_FIFO full   (0=No/ready, 1=Full/Busy) (10h words)          (R)
      Write: First round   (0=No change, 1=Reset BLKCNT and HASH)         (W)
1     Final round          (0=No/ready, 1=Enable/Busy)                  (R/W)
2     IN_FIFO DMA Enable   (0=Disable, 1=Enable CDMA DRQ 0Bh)           (R/W)
3     Byte order of Result (0=Little endian, 1=Big endian/Standard)     (R/W)
4-5   Mode                 (0=SHA256, 1=SHA224, 2=3=SHA1)               (R/W)
6-7   Unused (0)  ;reportedly "?" (but actually: always 0)                (?)
8     OUT_FIFO Enable      (0=No, 1=Readback Mode)           ;\optional (R/W)
9     OUT_FIFO Status      (0=Empty, 1=Non-empty)            ; readback   (R)
10    OUT_FIFO DMA Enable  (0=Disable, 1=Enable CDMA DRQ 0Ch);/          (R/W)
11-15 Unused (0)                                                          (-)
16-17 Unused (0)  ;reportedly "?" (but actually: always 0)                (?)
18-31 Unused (0)                                                          (-)
```

The optional readback mode allows to readback each 40h-byte block from the FIFO, this can reduce memory reads (and temporary memory writes), for example:

```
EMMC --> SHA --> AES --> Memory        ;saves 1xMemWrite and 2xMemRead
EMMC --> AES --> SHA --> Memory        ;saves 1xMemRead
```

The SHA_IN writing speed is about 62Mbyte/s for SHA256 and SHA224, and 50Mbyte/s for SHA1. Reading source data, and optional forwarding to SHA_OUT may cause extra slowdown.

1000A004h/10101004h - SHA_BLKCNT - SHA Input Length (R/W)

```
0-31  Length in bytes (0..FFFFFFFFh)
```

The length is automatically updated by hardware:

```
Length is reset to zero when setting SHA_CNT.bit0 (start).
Length increments by 40h after each 40h-byte FIFO block.
Length increments by remaining FIFO size after setting SHA_CNT.bit1 (final).
```

The hardware does automatically append the length value (and some padding bits) to the data stream before computing the final result.

1000A040h/10101040h - SHA_HASH - State/Result (20h bytes) (R/W)

Contains the SHA state/result. The word order is fixed, the byte order (per 32bit word) depends on SHA_CNT.bit3.

Setting SHA_CNT.bit0 does automatically apply the following initial values:

```
SHA256 6A09E667,BB67AE85,3C6EF372,A54FF53A,510E527F,9B05688C,1F83D9AB,5BE0CD19
SHA224 C1059ED8,367CD507,3070DD17,F70E5939,FFC00B31,68581511,64F98FA7,BEFA4FA4
SHA1   67452301,EFCDAB89,98BADCFE,10325476,C3D2E1F0,0       ,0       ,0
```

The values are updated after each 40h-byte FIFO block, and updated once more after final round.
SHA1 leaves the last 3 words unused (set to zero). SHA224 and SHA256 do internally use all 8 words (but the last word is usually omitted when reading the SHA224 result).

1000A080h/10301000h - SHA_FIFO (40h bytes) - SHA_IN (W) and SHA_OUT (R)

```
0-7   1st byte        ;\
8-15  2nd byte        ; data to be checksummed
16-23 3rd byte        ;
24-31 4th byte        ;/
```

The FIFO is mapped to a 40h-byte area at FIFO+0..3Fh. The word address is don't care (one can write all words to FIFO+0, or to FIFO+0,4,8,..,3Ch).
However, 8bit/16bit writes do REQUIRE the lower two addresss bits to match up with the number of previously written bytes (eg. byte writes must go to FIFO+0,1,2,3,4,5,..,3Fh or FIFO+0,1,2,3,0,1,.,3). Writing 8bit/16bit is normally needed only for the last block before setting final flag (and only if the length isn't a multiple of 40h). Writing 8bit/16bit may be also needed if the data comes from an odd source address (but that would slowdown everything).

Invalid Operations with/without Data Abort

```
Reading FIFO when CNT.bit8=0 returns ZERO (readback disabled)
Reading FIFO when CNT.bit8=1 and FIFO empty causes Data Abort (enabled+empty)
```

Untested...

```
Writing FIFO when FIFO full... is ignored? or data abort?
Writing 32bit to FIFO content is odd (not N*4 bytes)... causes what?
Writing 32bit to FIFO already contains 3Dh..3Fh bytes... causes what?
```

3DS Crypto - RSA Registers

RSA Registers (ARM9)

```
1000B000h 4    R/W RSA_CNT        Control/status and keyslot select
1000B0F0h 4    ?   RSA_UNKNOWN    Unknown
1000B1x0h 4    R/W RSA_SLOTCNT_x  Keyslot 0..3 control/status (x=0..3)
1000B1x4h 4    R   RSA_SLOTSIZE_x Keyslot 0..3 size/status    (x=0..3)
1000B200h 4    W   RSA_EXPFIFO    Exponent (10001h, or private key) ;\for
1000B204h FCh  W   RSA_EXPFIFO    Mirrors of above                  ; current
1000B400h 100h R/W RSA_MOD        Modulus (public key)              ;/keyslot
1000B800h 100h R/W RSA_DATA       Incoming Data and Result
```

1000B000h - ARM9 - RSA_CNT (R/W)

```
0     Start/Busy (0=Idle/Ready, 1=Enable/Busy)
1     IRQ Enable (0=Disable, 1=Enable, set ARM9 IF.bit22 when Ready)
2-3   Unused (0)
4-5   Keyslot    (0..3=Key 0-3)   ;for Start/Busy and RSA_MOD,EXPFIFO
6-7   Unused (0)
8     Byte order (0=Little endian, 1=Big Endian) ;for RSA_MOD,DATA,EXPFIFO
9     Word order (0=Little endian, 1=Big Endian) ;for RSA_MOD,DATA
10-31 Unused (0)
```

1000B0F0h - ARM9 - RSA_UNKNOWN (R and/or W)

```
0-28  Unknown/unused?   (always zero)
29    Unknown/readonly? (always set)
30-31 Unknown/unused?   (always zero)
```

The bootroms writes zero in RSA IRQ init, so some bits might be write-only? Writing seems to have no effect on anything though.

1000B100h+(0..3)*10h - ARM9 - RSA_SLOTCNT_0..3 (R/W)

```
0     Read: Exponent Status (0=Bad=LessThan4orOdd, 1=Good=4orMoreAndEven) (R)
      Write: Clear RSA_SLOTSIZE/EXPFIFO (0=Clear, 1=No Change)            (W)
1     Disable RSA_EXPFIFO Writes        (0=Normal, 1=DataAbort)         (R/W)
```

```
  2     Disable RSA_MOD Reads             (0=Normal, 1=DataAbort)         (R/W)
  3-30  Unused (0)
  31    Disable RSA_SLOTCNT_x Writes      (0=Normal, 1=Disable/permanent) (R/w)
```
Bit1,2 can be changed only if Status Bit0 was set (Good=4orMoreAndEven).

1000B104h+(0..3)*10h - ARM9 - RSA_SLOTSIZE_0..3 (R)
```
  0-31  Number of words written to EXPFIFO (range 0..40h)
```
Indicates the EXP size (and implied: the MOD and DATA sizes). Whilst that number is important, programmers should usually know how many words they had used, without needing to use the SLOTSIZE register.

1000B200h - ARM9 - RSA_EXPFIFO - Exponent; usually 10001h, or Private Key (W)
1000B204h..1000B2FFh - ARM9 - RSA_EXPFIFO mirrors (W)
```
  0-31  FIFO, in current byte-order, to be written MSW first (max 40h words)
```
The number of words written to EXPFIFO does imply the size of the MOD and DATA values (that makes sense for private exponents, but the public exponent 10001h must be padded with leading zeroes, and then followed by 10001h in last word (aka 01000100h for big-endian).
The RSA hardware wants the number of words to be an even number in range of 4..40h, aka a multiple of 8 bytes in range 10h..100h (and if so, sets SLOTCNT.bit0).

1000B400h..1000B4FFh - ARM9 - RSA_MOD - Modulus; Public Key (R/W)
```
  100h-byte area, in currently selected byte/word-order, for current keyslot
```
The upper bits are unused/don't care when SLOTSIZE<40h.

1000B800h..1000B8FFh - ARM9 - RSA_DATA (100h bytes) (R/W)
```
  100h-byte area, in currently selected byte/word-order
```
Contains the data that is to be encrypted/decrypted, and contains the result after completion.
The upper bits are unused/don't care when SLOTSIZE<40h.

RSA Timing Examples (versus 67MHz timer)
```
  0.9ms (E475h clks)   Public key, 80h-bytes (DSi-style)
  3.3ms (3571Dh clks)  Public key, 80h-bytes zeropadded to 100h-bytes size
  3.3ms (3574Ah clks)  Public key, 100h-bytes (3DS-style)
  200ms (CE59A5h clks) Private key, 100h-bytes (3DS-style)
```

Invalid Operations with/without Data Abort
```
  Reading MOD or DATA when busy (and maybe also on writing?)  --> Data Abort
  Reading MOD when disabled in SLOTCNT                        --> Data Abort
  Writing EXPFIFO when disabled in SLOTCNT                    --> Data Abort
  Writing more than 40h words to EXPFIFO                      --> Data Abort
  Reading EXPFIFO or reading unused registers like 1000B5xxh  --> Returns Zero
```

Keyslot 0..3 usage
During boot, the bootrom uses the following PUBLIC keys (100h byte modulus, with exponent 10001h):
```
  Slot 0 uninitialized                    (unused)
  Slot 1 retail=FFFFB1E0h, debug=FFFFC4E0h (for FIRM from eMMC)
  Slot 2 retail=FFFFB2E0h, debug=FFFFC5E0h (for FIRM from Wifi-Flash/NDS-Cart)
  Slot 3 retail=FFFFB0E0h, debug=FFFFC3E0h (for NCSD from eMMC)
```
After boot, the bootrom does replace the above keys by four PRIVATE keys (with 100h byte modulus, and 100h byte exponent):
```
  Slot 0 retail=FFFFB3E0h, debug=FFFFC6E0h ;\Hardware slots (modulus+exponent)
  Slot 1 retail=FFFFB5E0h, debug=FFFFC8E0h ; (the modulus are also stored in
  Slot 2 retail=FFFFB7E0h, debug=FFFFCAE0h ; RAM at ITCM+3D00h+(0..3)*100h)
  Slot 3 retail=FFFFB9E0h, debug=FFFFCCE0h ;/
  Slot 4 retail=FFFFBBE0h, debug=FFFFCEE0h ;\
  Slot 5 retail=FFFFBDE0h, debug=FFFFD0E0h ; RAM slots (modulus+exponent are
  Slot 6 retail=FFFFBFE0h, debug=FFFFD2E0h ; stored at ITCM+4100h+(0..3)*200h)
  Slot 7 retail=FFFFC1E0h, debug=FFFFD4E0h ;/
```
Later on, the firmware does replace some slots by other keys:
```
  Slot 0 Arbitrary (uh?)
```

```
  Slot 1 CXI access desc (following the exheader, uh?)
  Slot 2 Unused (contains the private key from bootrom)
  Slot 3 Unused (contains the private key from bootrom)
```

RSA Basics and Differences to DSi
The 3DS RSA hardware is more or less same as the DSi RSA BIOS functions, see there for general info about RSA maths and RSA padding:
BIOS RSA Functions (DSi only)
Apart from being hardware-implemented, the 3DS is usually using 100h-bytes (instead 80h), with SHA256 signatures (instead SHA1), and with OpenPGP headers (instead raw padding).

Blurb
Writing to RSA_MOD does not change the exponent written with RSA_EXPFIFO. An attack based on the Pohlig-Hellman algorithm exists to "read" the contents of RSA_EXPFIFO as a result (see 3DS System Flaws).

RSA Overview
The RSA module is essentially a hardware-accelerated modular exponentiation engine. It is specially optimized for RSA applications, so its behavior can be incoherent when RSA's invariants are broken.
The PKCS (Public-Key Cryptography Standards) message padding must be manually checked by software, as hardware will only do raw RSA operations.

Observed edge cases
```
  "if 2 divides mod, output == 0"
```
uh, how to "divide" a "mod" ?
uh, for mod2, remainder SHOULD be 0 or 1 (or is it ALWAYS 0 here?)
uh, also, MOD may be required to be bigger than DATA?

3DS Crypto - PRNG and OTP Registers

```
_____________________________ PRNG Registers _____________________________
```

PRNG Registers ... Pseudo Random Generator? Boot9, Process9
```
  10011000h 4  Random (changes on each read)
  10011010h 4  Random (changes on each read)
  10011020h 4  Random (constant)
```
Used as entropy-source for seeding random number generators.

```
_____________________________ OTP Registers _____________________________
```

OTP Registers
```
  10012000h 100h  Encrypted 3DS OTP  (R)
  10012100h 8     DSi OTP Console ID (R/W?)
```
Access to this region is disabled once the ARM9 writes 02h to CFG9_SYSPROT9.

10012000h - 100h bytes - Encrypted 3DS OTP (R)
Console-unique data encrypted with AES-CBC. The normalkey and IV are stored in Boot9 (retail/devunit have seperate normalkey+IV for this).

10012100h - 8 bytes - DSi OTP Console ID (R/W?)
Before writing CFG9_SYSPROT9 bit1, the ARM9 copies the 8-byte TWL Console ID here. This sets the registers at 4004D00h for ARM7.
```
  [10012100h] = DecryptedOTP[008h] xor B358A6AFh or 80000000h
  [10012104h] = DecryptedOTP[00Ch] xor 08C267B7h
```

Decrypted OTP

```
000h 90h   Copied into ITCM. The encrypted version of this is what
             New3DS-arm9loader hashes for key-generation.
000h 4     Always DEADB00Fh
004h 4     u32 DeviceId
008h 10h   Fall-back keyY used for movable.sed keyY when
             movable.sed doesn't exist in NAND (the last(???) two words
             here are used on retail for generating console-unique TWL
             keydata/etc).
             This is also used for "LocalFriendCodeSeed", etc.
018h 1     CTCert byte order? (usually 05h)
019h 1     CTCert issuer type:
             zero = retail  ("Nintendo CA - G3_NintendoCTR2prod")
             non-zero = dev ("Nintendo CA - G3_NintendoCTR2dev")
01Ah 6     Manufacturing date (year-1900, month, day, hour, minute, second)
020h 4     CTCert ECDSA exponent (byte-swapped when OTP[018h]>=5)
024h 2     Zero?
026h 1Eh   CTCert ECDSA private key
044h 3Ch   CTCert ECDSA signature
080h 10h   Zerofilled
090h 1Ch   Seed 1 for AES keys ;<-- used for console-unique AES keys
0ACh 1Ch   Seed 2 for AES keys ;\
0C8h 1Ch   Seed 3 for AES keys ; not actually used  ;\overlaps SHA256
0E4h 1Ch   Seed 4 for AES keys ;/                   ;/(see below)
0E0h 20h   SHA256 hash across [000h..0DFh]
```

OTP Blurp
After decryption, the first 90h-bytes of plaintext are copied to ITCM+3800h at 01FFB800h aka 07FFB800h if hash verification passes.

On FIRM versions prior to 3.0.0-X, this region was left unprotected. On versions since 3.0.0-X, this has been fixed, and the region disable is now done by Kernel9 after doing console-unique TWL keyinit, by setting bit 1 of CFG9_SYSPROT9. However, with the New3DS FIRM ARM9 binary this is now done in the FIRM ARM9 binary loader, which also uses the 10012000h region for New 3DS key generation.

On development units (UNITINFO != 0) ARM9 uses the first 8-bytes from 10012000h for the TWL Console ID. This region doesn't seem to be used by NATIVE_FIRM on retail at all, besides New3DS key-generation in the ARM9-loader.

3DS Crypto - AES Key Generator

Decrypt OTP and get otp_seed (and rom_seed for retail/debug)
```
if [10010010h]=00h then otp_keyiv=FFFFD6E0h else otp_keyiv=FFFFD700h
if [10010010h]=00h then rom_seed=FFFFD860h else rom_seed=FFFFDC60h
aes_setkey_normal(key=otp_keyiv+00h)
aes_CBC_decrypt(iv=otp_keyiv+10h, src=10012000h, dst=decrypted_otp, len=100h)
sha256(src=decrypted_otp, dst=temp, len=E0h)
if [temp+0..1Fh]=[decrypted_otp+E0h..FFh] then otp_okay=1 else otp_okay=0
if otp_okay=1 then otp_seed=decrypted_otp+90h else otp_seed=10012000h
if otp_okay=1 then copy [decrypted_otp+0..8Fh] to [ITCM+3800h..388Fh]
```

Generate bootrom keys (using above otp_seed and rom_seed)
```
sha256(src=otp_seed(00h..1Bh)+rom_seed(00h..23h), dst=temp, len=40h=1Ch+24h)
aes_setkey_xy(key_x=temp+00h, key_y=temp+10h)
aes_CBC_encrypt(iv=rom_seed+24h, src=rom_seed+34h, dst=temp, len=40h)
key_x.00h-03h = uninitialized
key_x.04h-07h = temp+00h
key_x.08h-0Bh = temp+10h
key_x.0Ch-0Fh = temp+20h
key_x.10h     = temp+30h
```

```
 key_x.11h-13h = uninitialized
;sha256(src=otp_seed(1Ch..37h)+rom_seed(74h..97h), dst=temp, len=40h=1Ch+24h)
;DO NOT: aes_setkey_xy(key_x=temp+00h, key_y=temp+10h)
 aes_CBC_encrypt(iv=rom_seed+98h, src=rom_seed+A8h, dst=temp, len=40h)
 key_x.14h     = temp+00h
 key_x.15h     = temp+10h
 key_x.16h     = temp+20h
 key_x.17h     = temp+30h
;sha256(src=otp_seed(38h..53h)+rom_seed(B8h..DBh), dst=temp, len=40h=1Ch+24h)
;DO NOT: aes_setkey_xy(key_x=temp+00h, key_y=temp+10h)
 aes_CBC_encrypt(iv=rom_seed+DCh, src=rom_seed+ECh, dst=temp, len=40h)
 key_x.18h-1Bh = temp+00h
 key_x.1Ch-1Fh = temp+10h
 key_x.20h-23h = temp+20h
 key_x.24h     = temp+30h
 key_x.25h-27h = uninitialized
;sha256(src=otp_seed(54h..6Fh)+rom_seed(12Ch..14Fh),dst=temp, len=40h=1Ch+24h)
;DO NOT: aes_setkey_xy(key_x=temp+00h, key_y=temp+10h)
 aes_CBC_encrypt(iv=rom_seed+150h, src=rom_seed+160h, dst=temp, len=40h)
 key_x.28h     = temp+00h
 key_x.29h     = temp+10h
 key_x.2Ah     = temp+20h
 key_x.2Bh     = temp+30h
 key_x.2Ch-2Fh = rom_seed+170h         ;\
 key_x.30h-33h = rom_seed+180h         ;
 key_x.34h-37h = rom_seed+190h         ;
 key_x.38h-3Bh = rom_seed+1A0h         ; fixed key X values
 key_x.3Ch     = rom_seed+1B0h         ;
 key_x.3Dh     = rom_seed+1C0h         ;
 key_x.3Eh     = rom_seed+1D0h         ;
 key_x.3Fh     = rom_seed+1E0h         ;/
 key_y.00h-03h = uninitialized
 key_y.04h     = rom_seed+1F0h         ;\
 key_y.05h     = rom_seed+200h         ;
 key_y.06h     = rom_seed+210h         ;
 key_y.07h     = rom_seed+220h         ; fixed key Y values
 key_y.08h     = rom_seed+230h         ;
 key_y.09h     = rom_seed+240h         ;
 key_y.0Ah     = rom_seed+250h         ;
 key_y.0Bh     = rom_seed+260h         ;/
 key_y.0Ch-3Eh = uninitialized
 key_y.3Fh     = as set for above CBC encrypt
 key.00h-03h   = uninitialized
 key.04h-0Bh   = uninitialized (or are they set via above key X/Y combo?)
 key.0Ch-0Fh   = rom_seed+270h         ;\
 key.10h       = rom_seed+280h         ;
 key.11h-13h   = uninitialized         ;
 key.14h       = rom_seed+290h         ;
 key.15h       = rom_seed+2A0h         ;
 key.16h       = rom_seed+2B0h         ;
 key.17h       = rom_seed+2C0h         ;
 key.18h-1Bh   = rom_seed+2D0h         ; fixed normal key values
 key.1Ch-1Fh   = rom_seed+2E0h         ;
 key.20h-23h   = rom_seed+2F0h         ;
 key.24h       = rom_seed+300h         ;
 key.25h-27h   = uninitialized         ;
 key.28h       = rom_seed+300h, too    ;
 key.29h       = rom_seed+310h         ;
 key.2Ah       = rom_seed+320h         ;
 key.2Bh       = rom_seed+330h         ;
 key.2Ch-2Fh   = rom_seed+340h         ;
 key.30h-33h   = rom_seed+350h         ;
 key.34h-37h   = rom_seed+360h         ;
 key.38h       = rom_seed+370h         ;
 key.39h-3Bh   = uninitialized         ;
```

```
key.3Ch       = rom_seed+370h, too    ;
key.3Dh       = rom_seed+380h         ;
key.3Eh       = rom_seed+390h         ;
key.3Fh       = rom_seed+3A0h         ;/
[FFF00618h]   = rom_seed+3A0h, too    ;-copy of normal key 3Fh
[temp] = fillword(rom_seed+3B0h)      ;-dummy/fill temp buffer on stack
```

Further important keys (changed after init by bootrom):

```
key_y.05h = 4D,80,4F,4E,99,90,19,46,13,A2,04,AC,58,44,60,BE  ;by New3DS FIRM
key_y.30h = movable.sed[110h..11Fh]
key_y.34h = movable.sed[110h..11Fh]
key_y.3Ah = movable.sed[110h..11Fh]
```

3DS Crypto - RSA sighax

sighax is a set of fake RSA signatures which are causing the bootrom to always pass RSA checks.

How it works
derrek's sighax relies on some bugs in the bootrom RSA signature verification. RSA doesn't throw errors when decrypting a wrong signature with wrong key (instead, it does just output "random" gibberish). The decrypted signature should normally consist of leading FFh-padding bytes, a header, and a SHA checksum.
However, the 3DS bootrom doesn't insist on the padding bytes, and it does ignore most of the header bytes. Finally, it does allow the header size to be as large as occupying the whole 100h-byte signature area, which results in the SHA value being read from next higher address AFTER the signature area - and the bootrom happens to store the computed SHA at that location (resulting the bootrom to compare the computed SHA against itself, instead of against of what should have been decrypted SHA value).
Alltogether, brute-forcing requires only 6 specific bytes in the decrypted signature (and the remaining 250 bytes are don't care).

The bootrom uses three RSA keys for different retail signatures, and another three RSA keys for debug version.

NCSD Signature (in eMMC)
The eMMC's NCSD Header is unencrypted, and can be changed without needing any AES keys (however, booting custom code requires to change FIRM, not NCSD).

```
6CF52F89F378120BFA4E1061D7361634D9A254A4F57AA5BD9F2C30934F0E68CB ;\
E6611D90D74CAAACB6A995565647333DC17092D320131089CCCD6331CB3A595D ;
1BA299A32FF4D8E5DD1EB46A2A57935F6FE637322D3BC4F67CFED6C2254C089C ; retail
62FA11D0824A844C79EE5A4F273D46C23BBBF0A2AF6ACADBE646F46B86D1289C ; version
7FF7E816CFDA4BC33DFF9D175AC69F72406C071B51F45A1ACB87F168C177CB9B ;
E6C392F0341849AE5D510D26EEC1097BEBFB9D144A1647301BEAF9520D22C55A ;
F46D49284CC7F9FBBA371A6D6E4C55F1E536D6237FFF54B3E9C11A20CFCCAC0C ;
6B06F695766ACEB18BE33299A94CFCA7E258818652F7526B306B52E0AED04218 ;/
53CB0E4EB1A6FF84284BE0E7385AB4A686A8BBCBC16102479280E0583655D271 ;\
3FE506FAEE74F8D10F1220441CC2FF5D6DDE99BE79C19B386CAF68D5EB8CED1A ;
AB4D243C5F398680D31CD2E3C9DD5670F2A88D563B8F65F5B234FD2EBB3BE44A ; debug
3B6C302722A2ADFB56AE3E1F6417BDEC1E5A86AABBAFBE9419ACA8FDCD45E2CD ; version
F1EB695F6EA87816122D7BE98EEF92C0814B16B215B31D8C813BB355CEA8138F ;
B3BF2374246842CD91E1F9AAFF76878617CE02064777AEA0876A2C245C784341 ;
CDEE90D691745908A6FF9CE781166796F9F1238F884C84D6F1EEBB2E40B4BCA0 ;
0A7B1E913E0980D29FF6061D8AA944C663F2638127F7CCAB6FC71538471A5138 ;/
```

FIRM Signature (in eMMC)
The eMMC's FIRM Header is encrypted via console-specific AES keys. Patching this area requires access to the corresponding AES keyslot (or knowing the OTP seed value that was used to generate that key). Alternately, knowing the decrypted content of the currently installed firmware version will also help (then one can simply XOR the encrypted bytes by old & new unencrypted bytes).

```
B6724531C448657A2A2EE306457E350A10D544B42859B0E5B0BED27534CCCC2A ;\
```

```
4D47EDEA60A7DD99939950A6357B1E35DFC7FAC773B7E12E7C1481234AF141B3 ;
1CF08E9F62293AA6BAAE246C15095F8B78402A684D852C680549FA5B3F14D9E8 ; retail
38A2FB9C09A15ABB40DCA25E40A3DDC1F58E79CEC901974363A946E99B4346E8 ; version
A372B6CD55A707E1EAB9BEC0200B5BA0B661236A8708D704517F43C6C38EE956 ;
0111E1405E5E8ED356C49C4FF6823D1219AFAEEB3DF3C36B62BBA88FC15BA864 ;
8F9333FD9FC092B8146C3D908F73155D48BE89D72612E18E4AA8EB9B7FD2A5F7 ;
328C4ECBFB0083833CBD5C983A25CEB8B941CC68EB017CE87F5D793ACA09ACF7 ;/
88697CDCA9D1EA318256FCD9CED42964C1E98ABC6486B2F128EC02E71C5AE35D ;\
63D3BF1246134081AF68754787FCB922571D7F61A30DE4FCFA8293A9DA512396 ;
F1319A364968464CA9806E0A52567486754CDDD4C3A62BDCE255E0DEEC230129 ; debug
C1BAE1AE95D786865637C1E65FAE83EDF8E7B07D17C0AADA8F055B640D45AB0B ; version
AC76FF7B3439F5A4BFE8F7E0E103BCE995FAD913FB729D3D030B2644EC483964 ;
24E0563A1B3E6A1F680B39FC1461886FA7A60B6B56C5A846554AE648FC46E30E ;
24678FAF1DC3CEB10C2A950F4FFA2083234ED8DCC3587A6D751A7E9AFA061569 ;
55084FF2725B698EB17454D9B02B6B76BE47ABBE206294366987A4CAB42CBD0B ;/
```

FIRM Signature (in WifiFlash or NtrCard)
These FIRM Headers are encrypted via fixed AES keys, and can be thus generated without knowing console-specific keys. However, there are some hardware requirements:
WifiFlash must have the write-protect pin disabled, and offset 400h must be writeable (this does probably (?) require a SPI flash chip with 128Kbyte or more; and won't work with the normal built-in chip). And, the eMMC firm sections must be 'destroyed' to trigger WifiFlash booting.
NtrCard requires a NDS flashcart, and support for rewriting its "bootmenu" memory region. And, key combination Start+Select+X with shell closed is needed to trigger NtrCard booting.

```
37E96B10BAF28C74A710EF35824C93F5FBB341CEE4FB446CE4D290ABFCEFACB0 ;\
63A9B55B3E8A65511D900C5A6E9403AAB5943CEF3A1E882B77D2347942B9E9EB ;
0D7566370F0CB7310C38CB4AC940D1A6BB476BCC2C487D1C532120F1D2A37DDB ; retail
3E36F8A2945BD8B16FB354980384998ECC380CD5CF8530F1DAD2FD74BA35ACB9 ; version
C9DA2C131CB295736AE7EFA0D268EE01872EF033058ABA07B5C684EAD60D76EA ;
84A18D866307AAAAB764786E396F2F8B630E60E30E3F1CD8A67D02F0A88152DE ;
7A9E0DD5E64AB7593A3701E4846B6F338D22FD455D45DF212C5577266AA8C367 ;
AE6E4CE89DF41691BF1F7FE58F2261F5D251DF36DE9F5AF1F368E650D576810B ;/
18722BC76DC3602E2C0171F3BCA12AB40EA6D112AEFBECF4BE7A2A58FF759058 ;\
A93C95CDA9B3B676D09A4E4C9E842E5C68229A6A9D77FAC76445E78EB5B363F8 ;
C66B166BE65AFAE40A1485A364C2C13B855CEEDE3DFEACEC68DD6B8687DD6DF8 ; debug
B6D3213F72252E7C03C027EE6079F9C5E0290E5DB8CA0BBCF30FCAD72EB637A1 ; version
70C4A2F41D96BF7D517A2F4F335930DC5E9792D78EDFB51DC79AD9D7A4E7F1ED ;
4D5A5C621B6245A7F1652256011DC32C49B955304A423009E2B78072CEBC12B3 ;
85B72F926F19318D64075F09278FBA8448FD2484B82654A55D064542A8F5D9F9 ;
828CDA5E60D31A40CF8EF18D027310DA4F807988BC753C1EB3B3FC06207E84DE ;/
```

SIGHAX.BIN
This signature is for unknown purpose (probably Retail eMMC FIRM, too):

```
6EFF209C8F4AF61F062413D602CA6B4DA1EB5AB9B6F1A2AB226A711DA2CCC27C
74DE1741143BF69058284CAF444F92A45AAFD5A068043323D48AF1D0EC05564E
BC79B55134E91A86C3788C97BC29D5A58A8A45255843B89122C7804542F72677
C8DA5EB7429BAF18F7A8B02E8BB940FE990E9DC97EDCF49DDB18092C28206E74
6753CC7C6E92362AA8D546B38D9E8D4311A6B1930DA14897807E304B5E1EC085
6EEFD62CEAEEF28B08BD80397A181560AE6FCE39D09C39DC3DED8C870AB6ABCE
2894940C0E9C4174F0131A0DA0747C4A7A42C9EC3487F109E252B7A9B865AE47
7895E8D6A42A0717C40BCCC7A735F33B1E3766AB0E4B5D681BAB410734AB62B0
```

Links & Credits
http://www.sighax.com/ ;one signature, and some general info
http://gist.github.com/SciresM/cdd2266efb80175d37eabbe86f9d8c52 ;six signatures

3DS DMA Registers

ARM9 NDMA

3DS DMA - NDMA Registers

ARM9 XDMA and ARM11 CDMA
3DS DMA - Corelink DMA Peripheral IDs
3DS DMA - Corelink DMA Register Summary
3DS DMA - Corelink DMA - Interrupt and Fault Status Registers
3DS DMA - Corelink DMA - Internal State Status Registers
3DS DMA - Corelink DMA - Transfer Start (aka "Debug" Registers)
3DS DMA - Corelink DMA - Fixed Configuration and ID Registers
3DS DMA - Corelink DMA Opcode Summary
The ARM Corelink DMA hardware is documented in
```
DDI0424D_dma330_r1p2_trm        - DMA controller (also covers r0p0 and r1p1)
IHI0022D amba axi_specification - Memory (ARCACHE, ARPROT etc.)
```

ARM11 GPU DMA
3DS GPU External Registers - Memfill and Memcopy
The GPU does probably also have DMA(s) for reading things like command lists, and texture data, and for rendering to framebuffer.

3DS DMA - NDMA Registers

NDMA
10002000h ARM9
Same as DSi, except, there seem to be 8 channels (instead of 4).
DSi New DMA (NDMA)
And with new bit in NDMAGCNT register:
```
0       SAD/DAD/TCNT/WCNT register read (0=Written value, 1=Current value)
```
And with new bits in NDMAxCNT registers:
```
0-4     New extra startup mode bits (used if startup=0Fh)
5-9     Unknown (unused on DSi, seem to be always 0 on 3DS, too)
```
And with changed startup modes:
```
00h     TIMER0                                          (Port 10003000h)
01h     TIMER1                                          (Port 10003004h)
02h     TIMER2                                          (Port 10003008h)
03h     TIMER3                                          (Port 1000300Ch)
04h     CTRCARD0                                       (maybe 10004000h)
05h     CTRCARD1                                       (maybe 10005000h)
06h     SDMMC controller (eMMC and SD/MMC slot)         (Port 10006000h)
07h     SDxx ?     ... apparently not SDIO wifi?      (maybe 10007000h)
08h     AES IN (WRFIFO)                                 (Port 10009008h)
09h     AES OUT (RDFIFO)                                (Port 1000900Ch)
0Ah     SHA IN (WRFIFO)                                 (Port 1000A000h)
0Bh     SHA OUT (RDFIFO) (optional readback)            (Port 1000A000h)
0Ch     NTRCARD      (used so by bootrom)              (maybe 10164000h)
0Dh     ?
0Eh     ?
0Fh     See below (subclassed)
10h-1Fh Start immediately (without repeat)
```
Mode 0Fh is subclassed by new bits from NDMAxCNT.bit0-4:
```
0Fh.00h     exists, related to SPI_CARD and AES IN     ;guess:
0Fh.01h     exists, related to SPI_CARD and AES IN     ;maybe these also
0Fh.02h     exists, related to SPI_CARD and AES OUT    ;work with CTRCARD0/1
0Fh.03h     exists, related to SPI_CARD and AES OUT
0Fh.04h     exists, related to SPI_CARD and SHA IN
0Fh.05h     exists, related to SPI_CARD and SHA IN
0Fh.06h     exists, related to SPI_CARD and SHA OUT
0Fh.07h     exists, related to SPI_CARD and SHA OUT
0Fh.08h     SDMMC DATA32 to AES IN  ?    ;used by bootrom
```

```
  0Fh.09h     ?   <-- maybe as above for 2nd SDxx controller?
  0Fh.0Ah     exists related to SDMMC and AES OUT
  0Fh.0Bh     ?
  0Fh.0Ch     exists related to SDMMC and SHA IN  ;(SHA at time of SDMMC!)
  0Fh.0Dh     ?
  0Fh.0Eh     exists related to SDMMC and SHA OUT
  0Fh.0Fh     ?
  0Fh.10h     AES OUT to SHA IN       ?    ;used by bootrom
  0Fh.11h     exists related AES IN and SHA OUT
  0Fh.12h-1Fh ?
```

Obviously still missing is MIC microphone (which should be supported by either NDMA, XDMA, or CDMA) (but maybe needs to be enabled somewhere, maybe selecting ARM/TEAK as target?).
Also nice would be having SDIO-Wifi on ARM9 side somewhere. Old NDS-Wifi might exist somewhere, too?

3DS DMA - Corelink DMA Peripheral IDs

CDMA Peripheral IDs (ARM11)

```
  00h                   Microphone (requires 1014010Ch.bit0=1)
  01h                   NTRCARD    (requires 1014010Ch.bit1=1)
  02h   camera (cam)    Camera Bus 0 (DSi cameras)     (Port 10120000h)
  03h   camera (cam)    Camera Bus 1 (left-eye camera) (Port 10121000h)
  04h   nwm             maybe wifi controller?
  05h   nwm             maybe debug wifi?
  06h   camera (y2r)    Y2R_0 INPUT_Y      ;\      (Port 10302000h)
  07h   camera (y2r)    Y2R_0 INPUT_U      ;       (Port 10302080h)
  08h   camera (y2r)    Y2R_0 INPUT_V      ; Y2R   (Port 10302100h)
  09h   camera (y2r)    Y2R_0 INPUT_YUV    ;       (Port 10302180h)
  0Ah   camera (y2r)    Y2R_0 OUTPUT_RGBA  ;/      (Port 10302200h)
  0Bh   fs              HASH (SHA FIFO IN) ;\SHA   (Port 10101000h)
  0Ch                   HASH (SHA FIFO OUT);/      (Port 10101000h)
  0Dh   TwlBg           LGYFB_0            ;\GBA   (Port 10110000h)
  0Eh   TwlBg           LGYFB_1            ;/NDS   (Port 10111000h)
  0Fh                   SPI_BUS0 (FIFO)    ;\      (Port 10160000h)
  10h                   SPI_BUS1 (FIFO)    ; SPIs  (Port 10142000h)
  11h                   SPI_BUS2 (FIFO)    ;/      (Port 10143000h)
  12h   mvd (y2r2)      Y2R_1 INPUT_Y      ;\    ;\
  13h   mvd (y2r2)      Y2R_1 INPUT_U      ;     ;
  14h   mvd (y2r2)      Y2R_1 INPUT_V      ; Y2R ; new CDMA controller
  15h   mvd (y2r2)      Y2R_1 INPUT_YUV    ;     ; in New3DS only
  16h   mvd (y2r2)      Y2R_1 OUTPUT_RGBA  ;/    ;
  17h   mvd             L2B_0 INPUT        ;\L2B ;
  18h   mvd             L2B_0 OUTPUT       ;/    ;
  19h   mvd             L2B_1 INPUT        ;\L2B ;
  1Ah   mvd             L2B_1 OUTPUT       ;/    ;
  1Bh                                            ;
  1Ch                                            ;
  1Dh                                            ;
  1Eh                                            ;
  1Fh                                            ;/
```

New3DS can use both Old CDMA and NewDMA, however, the peripheral data requests are sent only to either one (as selected in Port 10140410h).
Apart from peripherals, concerning ROM/RAM memory:
- CDMA can probably access whatever ARM11 memory (maybe whatever or so?)
- CDMA might maybe use Virtual memory addresses instead of physical addresses?

XDMA Peripheral IDs (ARM9)

```
  00h   Process9        "CTRCARD"(CARD0?) or SPI_CARD (FIFO) (Port 1000D000h)
  01h                   ?  (maybe CARD1?) or SPI_CARD (FIFO) (Port 1000D000h)
  02h                   ?
  03h                   ?
```

```
  04h                  AES IN (WRFIFO)                     (Port 10009008h)
  05h                  AES OUT (RDFIFO)                    (Port 1000900Ch)
  06h   unused???      SHA FIFO IN                         (Port 1000A000h)
  07h   Process9       SHA FIFO OUT (optional readback)    (Port 1000A000h)
```

Apart from peripherals, concerning ROM/RAM memory:
- XDMA can access ARM9 memory (probably Main RAM, and AXI, and maybe others?)
- XDMA reportedly cannot access the ARM9 bootrom at all

3DS DMA - Corelink DMA Register Summary

Corelink DMA Engines

```
  1000C000h ARM9  XDMA CoreLink DMA-330 r0p0 (four channels)
  10200000h ARM11 CDMA CoreLink DMA-330 r0p0 (eight channels)
  10206000h ARM11 CDMA CoreLink DMA-330 r1p1 (eight channels) (New3DS only)
  10300000h ARM11 DMA region (aka FIFOs are here?)
```

The New3DS has two DMA controllers on ARM11 side (the r0p0 is used by bootrom, and, after bootrom, everything is usually done by r1p1).

DMAC Control Registers

```
  10xxx000h       DSR           DMA Manager Status Register (R)
  10xxx004h       DPC           DMA Manager Program Counter (R)
  10xxx008h-01Ch  -             Reserved
  10xxx020h       INTEN         Interrupt Enable Register (R/W)
  10xxx024h       INT_EVENT_RIS Event-Interrupt Raw Status Register (R)
  10xxx028h       INTMIS        Interrupt Status Register (R)
  10xxx02Ch       INTCLR        Interrupt Clear Register (W)
  10xxx030h       FSRD          DMA Manager Fault Status Register (R)
  10xxx034h       FSRC          DMA Channel(s) Fault Status Register (R)
  10xxx038h       FTRD          DMA Manager Fault Type Register (R)
  10xxx03Ch       -             Reserved
  10xxx040h+n*4   FTRn          DMA channel n Fault type (R)
  10xxx060h-0FCh  -             Reserved
```

DMAC Channel Thread Status Registers

```
  10xxx100h+n*8   CSRn          DMA channel n Channel status (R)
  10xxx104h+n*8   CPCn          DMA channel n Channel PC (R)
  10xxx140h-3FCh  -             Reserved
```

DMAC AXI Status and Loop Counter Registers

```
  10xxx400h+n*20h SARn          DMA channel n Source address (R)
  10xxx404h+n*20h DARn          DMA channel n Destination address (R)
  10xxx408h+n*20h CCRn          DMA channel n Channel control (R)
  10xxx40Ch+n*20h LC0_n         DMA channel n Loop counter 0 (R)
  10xxx410h+n*20h LC1_n         DMA channel n Loop counter 1 (R)
  10xxx414h+n*20h -             Reserved
  10xxx418h+n*20h -             Reserved
  10xxx41Ch+n*20h -             Reserved
  10xxx500h-CFCh  -             Reserved
```

DMAC Debug Registers (debugging is meant to be synonym for "start execution")

```
  10xxxD00h       DBGSTATUS     Debug Status Register (R)
  10xxxD04h       DBGCMD        Debug Command Register (W) (start INST code)
  10xxxD08h       DBGINST0      Debug Instruction-0 (W) (code[2]+ch+thread)
  10xxxD0Ch       DBGINST1      Debug Instruction-1 (W) (byte[4])
  10xxxD10h-DFCh  -             Reserved
```

DMAC Configuration Registers

```
  10xxxE00h       CR0           Config Register 0, Misc Info (R)
  10xxxE04h       CR1           Config Register 1, Cache Info (R)
  10xxxE08h       CR2           Config Register 2, Boot Address (R)
  10xxxE0Ch       CR3           Config Register 3, Boot Event/Irq Security (R)
  10xxxE10h       CR4           Config Register 4, Boot Peripheral Security (R)
  10xxxE14h       CRD           DMA Configuration Register (R)
  10xxxE18h-E7Ch  -             Reserved
```

```
  10xxxE80h       WD            Watchdog Register (R/W) (rev1 only)
  10xxxE84h-FDCh  -             Reserved
```
Peripheral and component identification register
```
  10xxxFE0h-FECh  periph_id_n   Peripheral Identification Registers 0-3 (R)
  10xxxFF0h-FFCh  pcell_id_n    Component Identification Registers 0-3 (R)
```

CR0 values on 3DS
```
  [1000CE00h]=000167035h  ;ARM9         (12 events, 8 peripherals, 4 channels)
  [10200E00h]=0001f1075h  ;Old3DS/ARM11 (16 events, 18 peripherals, 8 channels)
  [10206E00h]=0003ff075h  ;New3DS/ARM11 (32 events, 32 peripherals, 8 channels)
```

3DS DMA - Corelink DMA - Interrupt and Fault Status Registers

_____ DMAC Control Registers _____

10xxx000h - DSR - DMA Manager Status Register (R)
```
  0-3   DMA Manager State (see below)
  4-8   DMA Manager DMA_WFE opcode (wait for event) (0..31 = Event 0..31)
  9     DMA Manager Non-Secure state (0=Secure, 1=Non-secure)
  10-31 Unused (undef)
```
DMA Manager States:
```
  00h     = Stopped
  01h     = Executing
  02h     = Cache Miss
  03h     = Updating PC
  04h     = Wait for event (WFE, see bit4-8 for event number)
  05h-0Eh = Reserved
  0Fh     = Faulting
```

10xxx004h - DPC - DMA Manager Program Counter (R)
```
  0-31  DMA Manager Program Counter
```

10xxx020h - INTEN - Interrupt Enable Register (R/W)
```
  0-31  Mode for DMA_SEV opcode event 0-31 "event_irq_select for event 0..31"
```
Bit [N] = 0 If the DMAC executes DMASEV for the event-interrupt resource N then the DMAC signals event N to all of the threads. Set bit [N] to 0 if your system design does not use irq[N] to signal an interrupt request.
Bit [N] = 1 If the DMAC executes DMASEV for the event-interrupt resource N then the DMAC sets irq[N] HIGH. Set bit [N] to 1 if your system design requires irq[N] to signal an interrupt request.
Note: See DMASEV on page 4-14 for information about selecting an event number.

10xxx024h - INT_EVENT_RIS - Event-Interrupt Raw Status Register (R)
10xxx028h - INTMIS - Interrupt Status Register (R)
```
  0-31  Event/Interrupt 0-31 Active (0=Inactive/Low, 1=Active/High)
```

10xxx02Ch - INTCLR - Interrupt Clear Register (W)
```
  0-31  Interrupt 0-31 (0=No change, 1=Clear Interrupt)
```

10xxx030h - FSRD - DMA Manager Fault Status Register (R)
```
  0     DMA Manager Thread in Faulting state (0=No, 1=Fault)
  1-31  Unused (undef)
```

10xxx034h - FSRC - DMA Channel(s) Fault Status Register (R)
```
  0-7   Fault Status for channel 0-7 (0=No, 1=Fault)
  8-31  Unused (undef)
```

10xxx038h - FTRD - DMA Manager Fault Type Register (R)
```
  0     Undefined Instruction                                   (0=No, 1=Fault)
```

```
  1     Invalid Instruction Operand                              (0=No, 1=Fault)
  2-3   Unused (undef)
  4     DMA_GO with inappropriate security permissions           (0=No, 1=Fault)
  5     DMA_WFE or DMA_SEC with inappropriate security           (0=No, 1=Fault)
  6-15  Unused (undef)
  16    AXI Response on RRESP bus              (0=OKAY, 1=EXOKAY/SLVERR/DECERR)
  17-29 Unused (undef)
  30    DMA Abort program-counter location (0=System memory, 1=Debug interface)
  31    Unused (undef)
```

10xxx040h+n*4 - FTRn - DMA channel n Fault type (R)

```
  0     Undefined Instruction                                    (0=No, 1=Fault)
  1     Invalid Instruction Operand                              (0=No, 1=Fault)
  2-4   Unused (undef)
  5     DMA_WFE or DMA_SEC with inappropriate security           (0=No, 1=Fault)
  6     Non-secure tried DMAWFP,DMALDP,DMASTP,DMAFLUSHP          (0=No, 1=Fault)
  7     Non-secure channel tried to change CCR                   (0=No, 1=Fault)
  8-11  Unused (undef)
  12    MFIFO too small for DMALD/DMAST                          (0=No, 1=Fault)
  13    MFIFO lacks data for DMAST                               (0=No, 1=Fault)
  14-15 Unused (undef)
  16    AXI Response on RRESP bus, opcode      (0=OKAY, 1=EXOKAY/SLVERR/DECERR)
  17    AXI Response on BRESP bus, data.write  (0=OKAY, 1=EXOKAY/SLVERR/DECERR)
  18    AXI Response on RRESP bus, data.read   (0=OKAY, 1=EXOKAY/SLVERR/DECERR)
  19-29 Unused (undef)
  30    DMA Abort program-counter location (0=System memory, 1=Debug interface)
  31    DMA channel locked-up because of insufficient resources (0=No, 1=Fault)
```

3DS DMA - Corelink DMA - Internal State Status Registers

```
_____________________ DMAC Channel Thread Status Registers _____________________
```

10xxx100h+n*8 - CSRn - DMA channel n Channel status (R)

```
  0-3   Channel Status   (see below)
  4-8   Wakeup number    (0-31, channel is waiting for event/peripheral 0-31)
  9-13  Unused (undef)
  14    DMAWFP executed with single/burst operand set     (0=Single, 1=Burst)
  15    DMAWFP executed with/without periph operand set   (0=Without, 1=With)
  16-20 Unused (undef)
  21    Channel operates in Non-Secure state         (0=Secure, 1=Non-secure)
  22-31 Unused (undef)
```

Channel Status:

```
  00h     = Stopped
  01h     = Executing
  02h     = Cache miss
  03h     = Updating PC
  04h     = Waiting for event
  05h     = At barrier
  06h     = reserved
  07h     = Waiting for peripheral
  08h     = Killing
  09h     = Completing
  0Ah-0Dh = reserved
  0Eh     = Faulting completing
  0Fh     = Faulting
```

10xxx104h+n*8 - CPCn - DMA channel n Channel PC (R)

```
  0-31  DMA Channel's Program Counter

_____________________ DMAC AXI Status and Loop Counter Registers _____________________
```

10xxx400h+n*20h - SARn - DMA channel n Source address (R)
10xxx404h+n*20h - DARn - DMA channel n Destination address (R)
```
  0-31  DMA Channel's Source/Destination memory address
```

10xxx408h+n*20h - CCRn - DMA channel n Channel control (R)
```
  0     Source Burst type      (0=Fixed address, 1=Incrementing address)
  1-3   Source Burst size      (0-4 = 1,2,4,8,16 bytes per beat) (5-7=Reserved)
  4-7   Source Burst len       (0-15 = 1..16 data transfers)
  8     Source ARPROT.bit0 Privileged  (0=Unprivileged, 1=Privileged) ;\access
  9     Source ARPROT.bit1 Non-secure  (0=Secure, 1=Non-secure)       ; type
  10    Source ARPROT.bit2 Instruction (0=Data, 1=Instruction)        ;/
  11-13 Source ARCACHE.bit0-2         ;ARCACHE.bit3 is always low
  14    Dest Burst type (0=Fixed address, 1=Incrementing address)
  15-17 Dest Burst size (0-4 = 1,2,4,8,16 bytes per beat) (5-7=Reserved)
  18-21 Dest Burst len  (0-15 = 1..16 data transfers)
  22    Dest AWPROT.bit0 Privileged    (0=Unprivileged, 1=Privileged) ;\access
  23    Dest AWPROT.bit1 Non-Secure    (0=Secure, 1=Non-secure)       ; type
  24    Dest AWPROT.bit2 Instruction   (0=Data, 1=Instruction)        ;/
  25-27 Dest AWCACHE.bit0,1,3  ;AWCACHE.bit2 is always low
  28-30 Endian swap bytes  (0=Off) (1-4=Within 16,32,64,128 bit) (5-7=Reserved)
  31    Unused (undef)
```
Bit9,23 should/must be both set, else faults (depending on secure mode, or so).

10xxx40Ch+n*20h - LC0_n - DMA channel n Loop counter 0 (R)
10xxx410h+n*20h - LC1_n - DMA channel n Loop counter 1 (R)
```
  0-7   Remaining loop count, minus 1   (00h..FFh = Loop 1..256 times)
  8-31  Unused (undef)
```

3DS DMA - Corelink DMA - Transfer Start (aka "Debug" Registers)

```
_______________________________ DMAC Debug Registers _______________________________
        (debugging is meant to be a synonym for "start execution")
```

10xxxD00h - DBGSTATUS - Debug Status Register (R)
```
  0     Debug Status  (0=Idle, 1=Busy)    ;aka INST0/INST1 busy?
  1-31  Unused (undef)
```

10xxxD04h - DBGCMD - Debug Command Register (W) (start INST code)
```
  0-1   Execution (0=Execute DBGINST, 1..3=Reserved)
  2-31  Unused (0)
```

10xxxD08h - DBGINST0 - Debug Instruction-0 (W) (code[2]+ch+thread)
10xxxD0Ch - DBGINST1 - Debug Instruction-1 (W) (byte[4])
```
  0     Debug thread  (0=DMA Manager thread, 1=DMA channel 0..7) ;\
  1-7   Reserved (0)                                             ; usually
  8-10  DMA channel (0..7 for channel 0..7) (when above bit0=1)  ; zero
  11-15 Reserved (0)                                             ;/
  16-23 Instruction Byte 0    ;-usually DMA_GO opcode
  24-31 Instruction Byte 1    ;-usually target_channel for DMA_GO
  32-39 Instruction Byte 2    ;\
  40-47 Instruction Byte 3    ; usually target_address for DMA_GO
  48-55 Instruction Byte 4    ;
  56-63 Instruction Byte 5    ;/
```
Examples:
```
  db 00h,00h,DMA_GO,<chn>,<addr32bit>  ;<-- start channel code
  db 01h,<chn>,DMA_KILL,0,0,0,0,0      ;<-- kill channel (eg. after fault/hang)
```

3DS DMA - Corelink DMA - Fixed Configuration and ID Registers

```
_____________________ DMAC Configuration Registers _____________________
```

10xxxE00h - CR0 - Configuration Register 0 - Misc Info (R)

```
  0     Peripheral Requests supported         (0=None, 1=Yes, see bit12-16)
  1     boot_from_pc signal state at reset    (0=Was Low, 1=Was High)
  2     boot_manager_ns signal state at reset (0=Was Low, 1=Was High)
  3     Reserved (undef)
  4-6   Number of supported DMA channels      (0..7 = 1..8 channels)
  7-11  Reserved (undef)
  12-16 Number of supported Peripherals       (0..31 = 1..32 peripherals)
  17-21 Number of supported Interrupt outputs (0..31 = 1..32 interrupts)
  22-31 Reserved (undef)
```

Bit1/2 are probably related to CR2 boot address, excecuting in secure or non-secure state... but it's unspecified if Low/High means on or off.

```
  CR0 values on 3DS:
  [1000CE00h]=000167035h  ;ARM9         (12 events, 8 peripherals, 4 channels)
  [10200E00h]=0001f1075h  ;Old3DS/ARM11 (16 events, 18 peripherals, 8 channels)
  [10206E00h]=0003ff075h  ;New3DS/ARM11 (32 events, 32 peripherals, 8 channels)
```

10xxxE04h - CR1 - Configuration Register 1 - Cache Info (R)

```
  0-2   Size of an i-cache line (2,3,4,5 = 4,8,16,32 bytes) (0,1,6,7=Reserved)
  3     Reserved (undef)
  4-7   Number of i-cache lines (0..15 = 1..16 lines)
  8-31  Reserved (undef)
```

10xxxE08h - CR2 - Configuration Register 2 - Boot Address (R)

```
  0-31  Reset entrypoint (for DMA Manager?)
```

10xxxE0Ch - CR3 - Configuration Register 3 - Boot Event/Irq Security (R)

```
  0-31  Security state for Event-interrupt 0..31 (0=Secure, 1=Non-secure)
```

10xxxE10h - CR4 - Configuration Register 4 - Boot Peripheral Security (R)

```
  0-31  Security state for Peripheral 0..31      (0=Secure, 1=Non-secure)
```

10xxxE14h - CRD - DMA Configuration Register (R)

```
  0-2   AXI master Data width (2,3,4 = 32,64,128 bit) (0,1,5,6,7=Reserved)
  3     Reserved (undef)
  4-6   Write capability, max number outstanding transactions (0..7 = 1..8)
  7     Reserved (undef)
  8-11  Write queue depth                              (0..15 = 1..16 lines)
  12-14 Read capability, max number outstanding transactions  (0..7 = 1..8)
  15    Reserved (undef)
  16-19 Read queue depth                               (0..15 = 1..16 lines)
  20-29 Data buffer depth                           (0..3FFh = 1..1024 lines)
  30-31 Reserved (undef)
```

10xxxE80h - WD - Watchdog Register (R/W) (rev1 only)

Supported on rev1 only (ie. r1p0/r1p1/r1p2, but not r0p0).

```
  0     Action on Lock-up (0=IRQ and Abort contributing channels, 1=IRQ only)
  1-31  Reserved (undef)
```

```
_______________ Peripheral and component identification register _______________
```

10xxxFE0h-FECh - periph_id_n - Peripheral Identification Registers 0-3 (R)

This region contains four 32bit registers (30h,13h,x4h,00h). One is supposed to extract the lower 8bit of these 32bit values, and then to merge them into a "conceptual-32bit-value" (00x41330h). And then interprete it as so:

```
  0-11  Part number            (330h=DMAC)
```

```
  12-19 Designer               (41h=ARM)
  20-23 Revision               (0=r0p0, 1=r1p0, 2=r1p1, 3=r1p2)
  24    Integration test logic (0=None, 1=Exists)
  25-31 Reserved (undef)
```

10xxxFF0h-FFCh - pcell_id_n - Component Identification Registers 0-3 (R)
This region contains four 32bit registers (0Dh,F0h,05h,B1h). One is supposed to extract the lower 8bit of these 32bit values, and then to merge them into a "conceptual-32bit-value" (B105F00Dh). And then interprete it as so:

```
  0-31  Component ID (B105F00Dh)
```

3DS DMA - Corelink DMA Opcode Summary

Corelink DMA Opcode Summary

```
  Opcode             CM Syntax
  00h                CM DMAEND                  ;-done  (finish transfer)
  01h                CM DMAKILL                 ;-abort (unfinished transfer)
  04h                C- DMALD                   ;\
  05h                C- DMALDS           ;\cond ; load from SAR (to dmafifo)
  07h                C- DMALDB           ;/     ;/
  08h                C- DMAST                   ;\
  09h                C- DMASTS           ;\cond ; store to DAR (from dmafifo)
  0Bh                C- DMASTB           ;/     ;/
  0Ch                C- DMASTZ                  ;-store zero to DAR
  12h                C- DMARMB                  ;-read memory barrier
  13h                C- DMAWMB                  ;-write memory barrier
  18h                CM DMANOP                  ;-no operation (align padding)
  20h,len-1          C- DMALP lpc0,len          ;\loop start with loop count
  22h,len-1          C- DMALP lpc1,len          ;/
  25h,periph*8       C- DMALDPS periph   ;\     ;\load and notify peripheral
  27h,periph*8       C- DMALDPB periph   ; cond ;/
  29h,periph*8       C- DMASTPS periph   ;      ;\store and notify peripheral
  2Bh,periph*8       C- DMASTPB periph   ;/     ;/
  2Ch,rel_addr       C- DMALPEND                ;-loop end for DMALPFE
  30h,periph*8       C- DMAWFP periph,single    ;\
  31h,periph*8       C- DMAWFP periph,periph    ; wait for peripheral
  32h,periph*8       C- DMAWFP periph,burst     ;/
  34h,event*8        CM DMASEV event            ;-send event (or interrupt)
  35h,periph*8       C- DMAFLUSHP periph        ;-init peripheral
  36h,event*8+00h    CM DMAWFE event            ;\wait for event (with optional
  36h,event*8+02h    CM DMAWFE event,invalid    ;/  code cache invalidate)
  38h,rel_addr       C- DMALPEND  lpc0          ;\
  39h,rel_addr       C- DMALPENDS lpc0   ;\cond ;
  3Bh,rel_addr       C- DMALPENDB lpc0   ;/     ; loop end for DMALP lpc0/1
  3Ch,rel_addr       C- DMALPEND  lpc1          ;
  3Dh,rel_addr       C- DMALPENDS lpc1   ;\cond ;
  3Fh,rel_addr       C- DMALPENDB lpc1   ;/     ;/
  54h,imm16          C- DMAADDH SAR,imm16       ;\add halfword
  56h,imm16          C- DMAADDH DAR,imm16       ;/  ("positive")
  5Ch,imm16          C- DMAADNH SAR,imm16       ;\add imm16+0FFFF0000h
  5Eh,imm16          C- DMAADNH DAR,imm16       ;/  ("negative", rev1 only)
  A0h,channel,imm32  -M DMAGO channel,imm32     ;\jump/goto entrypoint
  A2h,channel,imm32  -M DMAGO channel,imm32,ns  ;/  (ns=non-secure mode)
  BCh,00h,imm32      C- DMAMOV SAR,imm32        ;\
  BCh,01h,imm32      C- DMAMOV CCR,imm32        ; move to register
  BCh,02h,imm32      C- DMAMOV DAR,imm32        ;/
  imm8               -- DCB imm8                ;\manually defined 'code'
  imm32              -- DCD imm32               ;/
  -  (no opcode)     C- DMALPFE   ;loop forever ;-loop start without loop count
```

All opcodes and parameters (except imm16/imm32) are 8bit wide. The CM column indicates if the opcode can be used in Channel or Manager threads (or both).
The rel_addr parameters are unsigned offsets (jump target is $-rel_addr).

Nocash Syntax
Allows MOV instead LP, and LOOP instead LPEND, and "ADD reg,+/-imm" instead of "ADDH/ADNH reg,imm".

3DS Config Registers

3DS Config - CONFIG9 Registers

CONFIG9 Registers

Address	Width	Old3DS	Name	Used by
10000000h	1	Yes	CFG9_SYSPROT9	Boot9
10000001h	1	Yes	CFG9_SYSPROT11	Boot9
10000002h	1	Yes	CFG9_RST11	Boot9
10000004h	4	Yes	CFG9_DEBUGCTL	
10000008h	1	Yes	CFG9_AES	Boot9, Process9, TwlProcess9
1000000Ch	2	Yes	CFG9_CARD_CTL	Process9
10000010h	1	Yes	CFG9_CARD_POWER	Process9
10000012h	2	Yes	CFG9_CARD_INSERT_DELAY	Boot9, Process9
10000014h	2	Yes	CFG9_CARD_PWROFF_DELAY	Boot9, Process9
10000020h	2	Yes	CFG9_SDMMCCTL	Process9
10000100h	2	Yes	CFG9_UNKNOWN	
10000200h	1	No	CFG9_EXTMEMCNT9 (New3DS)	NewKernel9
10000FFCh	4	Yes	CFG9_MPCORECFG	
10010000h	4	Yes	CFG9_BOOTENV	
10010010h	1	Yes	CFG9_UNITINFO	Process9
10010014h	1	Yes	CFG9_TWLUNITINFO	Process9

10000000h - CFG9_SYSPROT9

0	Disables ARM9 bootrom "(+8000h)" when set to 1, and enables access to FCRAM. Cannot be cleared to 0 once set to 1. Boot9
1	Disables OTP area when set to 1. Cannot be cleared to 0 once set to 1. NewKernel9Loader, Process9
2-7	Unused (0)

On Old 3DS, NATIVE_FIRM reads CFG9_SYSPROT9 to know whether it has previously initialized the TWL console-unique keys using the OTP data. After setting the TWL console-unique keys, NATIVE_FIRM sets CFG9_SYSPROT9 bit 1 to disable the OTP area. In subsequent FIRM launches prior to the next reset, NATIVE_FIRM will see that the OTP area is disabled, and skip this step.

On New 3DS, the above is instead done by the Kernel9 loader. In addition to using the OTP data for initializing the TWL console-unique keys, the Kernel9 loader will generate the decryption key for NATIVE_FIRM. The final keyslot for NATIVE_FIRM is preserved, so that at a non-reset FIRM launch, the keyslot can be reused, since the OTP would then be inaccessible.

10000001h - CFG9_SYSPROT11

0	Disables ARM11 bootrom "(+8000h)" when set to 1, and enables access to FCRAM. Cannot be cleared to 0 once set to 1. Boot9
1-7	Unused (0)

10000002h - CFG9_RST11

```
0     Presumably takes ARM11 out of reset. Cannot be set to 1 once it has
        been cleared.
      Reportedly: Bit0 is actually for "write-protecting the bootrom area"
1-7   Unused (0)
```

10000004h - CFG9_DEBUGCTL

```
0-31  Whatever, debug related (readonly, always zero on retail consoles)
```

10000008h - CFG9_AES (R/W)

```
0-1   Unknown (R/W)
2-3   AES related? Value 3 written after write to AES_CTL (R/W)
4-7   Unused (0)
```

1000000Ch - CFG9_CARD_CTL - Gamecard Controller Select (R/W)

```
0-1   Gamecard ROM controller  (0=NTRCARD, 1=?, 2=CTRCARD0, 3=CTRCARD1)
2-3   Unused (0)
4     Gamecard SPI_CARD mode   (0=Manual, 1=FIFO)
5-7   Unused (0)
8     Gamecard SPI controller  (0=NTRCARD, 1=SPI_CARD)
9-11  Unused (0)
12    Unknown...? (R/W)
13-15 Unused (0)
```

There are three controllers for ROM cartridge commands:

```
xxx0h/xxx1h? NTRCARD  (8-byte commands)  (bit1=0)         (Port 10164000h)
xxx2h        CTRCARD0 (16-byte commands) (bit1=1, bit0=0) (Port 10004000h)
xxx3h        CTRCARD1 (16-byte commands) (bit1=1, bit0=1) (Port 10005000h)
```

And three controllers for SPI-bus cartridge savedata:

```
x0x0h/x0x1h  NTRCARD Manual NDS-style    (bit8=0, bit1=0) (Port 10164000h)
x0x2h/x0x3h  None                        (bit8=0, bit1=1) (N/A)
x10xh        SPI_CARD in Manual Mode     (bit8=1, bit4=0) (Port 1000D000h)
x11xh        SPI_CARD in FIFO Mode       (bit8=1, bit4=1) (Port 1000D800h)
```

The deselected controllers are disconnected from the cartridge bus (and tend to return data=FFh when trying to read from the cartridge).

10000010h - CFG9_CARD_POWER (same as SCFG_MC on DSi) (R/W)

```
0     NDS Slot Game Cartridge (0=Inserted, 1=Ejected)              (R)
1     NDS Slot Unknown/Unused (0)
2-3   NDS Slot Power State (0=Off, 1=On+Reset, 2=On, 3=RequestOff)  (R/W)
4-15  Unused (0)
```

Same as SCFG_MC on DSi (but with the bits for 2nd cartridge slot removed). See DSi specs for cartridge power on/off sequences.

DSi Control Registers (SCFG)

10000012h - CFG9_CARD_INSERT_DELAY (usually 1988h = 100ms) (R/W)
10000014h - CFG9_CARD_PWROFF_DELAY (usually 264Ch = 150ms) (R/W)

Same as 4004012h/4004014h on DSi, see DSi SCFG registers for details.

```
0-15  Delay in 400h cycle units (at 67.027964MHz)  ;max FFFFh=ca. 1 second
```

There are related IRQs on ARM9 and ARM11 side (CGC aka Change Gamecard or so):

```
ARM9 IF.bit25, and ARM11 Interrupt 74h   ;at begin of PWROFF delay
ARM9 IF.bit26, and ARM11 Interrupt 75h   ;at end of INSERT delay
```

10000020h - CFG9_SDMMCCTL

Controls power? Also controls SD card detect?

```
0     SD slot power                                (1=Unknown?) Process9
1     eMMC power? NAND init hangs if set           (1=Set?)     Process9
2-3   Other power supplies?                        (??)         Process9
4-5   Unused (0)
6     Unknown, set at cold boot                    (1=Set?)     R/W
```

```
7     Unknown                                          (?)          R/W
8     ? This bit seems to do nothing.                  (?)          Process9
9     If not set force pulls all SD card lines high (0=Not set?) Process9
10-15 Unused (0)
```

10000100h - CFG9_UNKNOWN

```
0-1   Unknown (R/W)
2     Unused (0)
3     Unknown (R/W)
4-7   Unused (0)
8-13  Unknown (R/W)
14-15 Unused (0)
```

10000200h - New3DS - CFG9_EXTMEMCNT9

```
0     New3DS: Extended ARM9 memory at 08100000h..0817FFFFh (0=Off, 1=On)
1-31  Unused (0)
```

10000FFCh - CFG9_MPCORECFG (R)

Identical to "PDN_MPCORE_CFG". Uh, what?
Above seems to refer to one of the "CONFIG11" registers.
Perhaps 10140FFCh - 16bit CFG11_SOCINFO, though the ARM9 register is 32bit?
Or maybe one of the CFG11_MPCORE_xxx registers, though those are New3DS only?
Or maybe it does actually refer to ARM11 cp15 coprocessor registers???
Or maybe one of the MPCore registers at 17E0xxxxh?
PROBABLY it's CFG11_SOCINFO (as they are both read-only 0007h on New3DS)!

10010000h - CFG9_BOOTENV

```
0-31  Value (0..FFFFFFFFh) (R/W)
```

This register is used to determine what the previous running FIRM was. Its value is kept following an MCU reboot. Its initial value (on a cold boot) is 0. NATIVE_FIRM sets it to 1 on shutdown/FIRM launch.

```
LGY FIRM writes value 3 here when launching a TWL title
LGY FIRM writes value 7 when launching an AGB title
```

NATIVE_FIRM will only launch titles if this is not value 0, and will only save the AGB_FIRM savegame to SD if this is value 7.

10010010h - CFG9_UNITINFO (R)

This 8bit register is value zero for retail, non-zero for dev/debug units.

10010014h - CFG9_TWLUNITINFO (R/W)

```
0-1   Value (to be copied from CFG9_UNITINFO)
2-7   Unused (0)
```

In the console-unique TWL key-init/etc function the ARM9 copies the u8 value from REG_UNITINFO to this register.

3DS Config - CONFIG11 Registers

CONFIG11 Registers (also referred to as "PDN" Registers)

Address	Width	Old3DS	Name	Used by
10140000h	1*8	Yes	CFG11_SHAREDWRAM_32K_CODE<0-7>	Boot11,Process9,DSP
10140008h	1*8	Yes	CFG11_SHAREDWRAM_32K_DATA<0-7>	Boot11,Process9,DSP
10140100h	2	Yes	?	
10140102h	2	Yes	?	
10140104h	1	Yes	CFG11_FIQ_CNT	Kernel11
10140105h	1	Yes	?	Kernel11
10140108h	2	Yes	? Related to HID_PAD_IRQ (??)	TwlBg
1014010Ch	2	Yes	CFG11_CDMA_CNT	TwlBg
10140140h	4	Yes	CFG11_GPUPROT	Kernel11

```
10140180h 1     Yes     CFG11_WIFICNT                   TwlBg, NWM Services
101401C0h 4     Yes     CFG11_SPI_CNT                   SPI Services, TwlBg
10140200h 4     Yes     ?
10140400h 1     No      ?  Clock related?               NewKernel11
10140410h 4     No      CFG11_CDMA_PERIPHERALS          NewKernel11
10140420h 1     No      CFG11_BOOTROM_OVERLAY_CNT       NewKernel11
10140424h 4     No      CFG11_BOOTROM_OVERLAY_VAL       NewKernel11
10140428h 4     No      ?
10140FFCh 2     Yes     CFG11_SOCINFO                   Boot11, Kernel11
10141000h 2     Yes     CFG11_GPU_STATUS       ;???     Kernel11, TwlBg
10141008h 4     Yes     CFG11_PTM_0            ;?       PTM, PDN
1014100Ch 4     Yes     CFG11_PTM_1            ;?       PTM, TwlBg, PDN
10141100h 2     Yes     CFG11_TWLMODE_BOOT              TwlProcess9, TwlBg
10141104h 2     Yes     CFG11_TWLMODE_1        ;?       TwlBg
10141108h 2     Yes     CFG11_TWLMODE_2        ;?       TwlBg
1014110Ah 2     Yes     CFG11_TWLMODE_HID_IRQ           TwlBg
1014110Ch 1     Yes     CFG11_WIFIUNK                   NWM Services
10141110h 2     Yes     CFG11_TWLMODE_HID_MODE          TwlBg
10141112h 2     Yes     CFG11_TWLMODE_HID_SET           TwlBg
10141114h 2     Yes     CFG11_CODEC_0                   Codec, TwlBg
10141116h 2     Yes     CFG11_CODEC_1                   Codec, TwlBg
10141118h 1     Yes     ?                               TwlBg
10141119h 1     Yes     ?                               TwlBg
10141120h 1     Yes     ?                               TwlBg
10141200h 4     Yes     CFG11_GPU_CNT         Boot11, Kernel11, PDN, TwlBg
10141204h 4     Yes     CFG11_GPU_CNT2        Boot11, Kernel11, TwlBg
10141208h 1     ?       ? GPU related?        Boot11
10141210h 2     Yes     CFG11_GPU_FCRAM_CNT             Kernel11, TwlBg
10141220h 1     Yes     CFG11_CODEC_CNT                 Boot11, TwlBg, PDN
10141224h 1     Yes     CFG11_CAMERA_CNT                PDN Services
10141230h 1     Yes     CFG11_DSP_CNT                   Process9, PDN
10141240h ?     No      unknown (nonzero in New3DS mode) XXX !!!!!!!!!!!!
10141300h 4     No      CFG11_MPCORE_CLKCNT             NewKernel11
10141304h 2     No      CFG11_MPCORE_CNT                NewKernel11
10141310h 1     No      CFG11_MPCORE_BOOTCNT_CPU0       NewKernel11
10141311h 1     No      CFG11_MPCORE_BOOTCNT_CPU1       NewKernel11
10141312h 1     No      CFG11_MPCORE_BOOTCNT_CPU2       NewKernel11
10141313h 1     No      CFG11_MPCORE_BOOTCNT_CPU3       NewKernel11
```

10140000h+0..7 - CFG11_SHAREDWRAM_32K_CODE --- aka MBK
10140008h+0..7 - CFG11_SHAREDWRAM_32K_DATA --- aka MBK
Used for mapping 32K chunks of shared WRAM for Teak DSP code/data.

```
0     Master (0=ARM only, 1=ARM and DSP)
1     Unused (0)
2-4   Offset (0..7) (slot 0..7) (LSB of address in 32Kbyte units)
5-6   Unused (0)
7     Enable (0=Disable, 1=Enable)
```

ARM11 and ARM9 can access the memory regardless of bit0 (but trigger data abort if bit7 is cleared, or on missing slot values in bit2-4).

```
DSP Code is mapped at 1FF00000h..1FF3FFFFh in ARM memory (256Kbytes)
DSP Data is mapped at 1FF40000h..1FF7FFFFh in ARM memory (256Kbytes)
```

10140100h - ??? 1bit? (R/W)
10140102h - ??? 1bit? (R) -- But, cleared in New3DS mode?
Unknown.

10140104h - CFG11_FIQ_CNT

```
0-1   FIQ related? (R/W)
2-3   New3DS only: unknown...?      ;initially set, (R/W) in New3DS mode!
4-7   Unused (0)
```

"Writing bit1 to this register disables FIQ interrupts.
This bit is set upon receipt of a FIQ interrupt and when svcUnbindInterrupt is called on the FIQ-abstraction

software interrupt for the current core. It is cleared when binding that software interrupt to an event and just before that event is signaled."
Maybe bit0-3 are for CPU0-3 (with above mentioned bit1 being for CPU1)? If the bits are really FIQ related, then they are probably used on debug hardware only.

10140105h - ???

```
0-1   Unknown...? (R/W)
2-3   New3DS only: unknown...?      ;initially set, (R/W) in New3DS mode!
4-7   Unused (0)
```

Unknown. Maybe bit0-3 are for CPU0-3?

10140108h - Related to HID_PAD_IRQ (??) TwlBg

```
0     Unknown...? (R/W)
1-15  Unused (0)
```

1014010Ch - CFG11_CDMA_CNT (R/W)

```
0     Enable Microphone DMA             (0=Off, 1=Enable CDMA 00h)      (R/W)
1     Enable NTRCARD DMA on ARM11 side  (0=Off, 1=Enable CDMA 01h)      (R/W)
2-3   Unused (0)
4-5   Unknown...?                                                       (R/W)
6-15  Unused (0)
```

10140140h - CFG11_GPUPROT

```
0-3   Old FCRAM DMA cutoff size, 0 = no protection.
4-7   New FCRAM DMA cutoff size, 0 = no protection.  ;<--New3DS only
8     AXIWRAM protection, 0 = accessible.
9-10  QTM DMA cutoff size                          ;<--New3DS only
11-31 Unused (0)
```

When this register is set to value 0, the GPU can access the entire FCRAM, AXIWRAM, and on New3DS all QTM-mem. On cold boot this reg is set to 0.
Initialized during kernel boot, and used with SVC 59h which was implemented with v11.3.
The New3DS bits are writeable only after enabling New3DS mode.
For the old FCRAM DMA cutoff, it protects starting from 28000000h-(800000h*x) until end of FCRAM. There is no way to protect the first 800000h-bytes.
For the new FCRAM DMA cutoff, it protects starting from 30000000h-(800000h*x) until end of FCRAM.
When the old FCRAM cutoff is set to non-zero, the first 800000h-bytes bytes of new FCRAM are protected.
On New3DS the old+new FCRAM cutoff can be used at the same time, however this isn't done officially.
For the QTM DMA cutoff, it protects starting from 1F400000h-(100000h*x) until end of QTM mem.

10140180h - CFG11_WIFICNT

```
0     Enable wifi subsystem
1-7   Unused (0)
```

101401C0h - CFG11_SPI_CNT

```
0     SPI_BUS0 Mode (registers at 10160000h)  (0=Manual, 1=Fifo/Autopoll)
1     SPI_BUS1 Mode (registers at 10142000h)  (0=Manual, 1=Fifo/Autopoll)
2     SPI_BUS2 Mode (registers at 10143000h)  (0=Manual, 1=Fifo/Autopoll)
3-31  Unused (0)
```

The registers for the deselected mode are disconnected from the bus, but they are still working internally (that is, transfer attempts will finish after the expected amount of time, but SPI reads return nothing useful: FFh for Manual reads, and 00h for FIFO/Autopoll reads).

10140200h - ???

```
0-31  Unknown (R/W)
```

Unknown.

10140400h - New3DS - Clock related?

```
0-1   Unknown (R/W, even if New3DS didn't enable New3DS mode)
```

```
  2-31  Zero
```

10140410h - New3DS - CFG11_CDMA_PERIPHERALS (R/W)
```
  0-17  CDMA Peripheral 00h-11h data request target (0=Old CDMA, 1=New CDMA)
  18-31 Zero
```

10140420h - New3DS - CFG11_BOOTROM_OVERLAY_CNT (R/W)
```
  0     ARM11 Bootrom Overlay Enable (0=Disable, 1=Enable)
  1-31  Unused (0)
```
When enabled, opcode/data reads from physical(?) address 00010000h-00010FFFh and FFFF0000h-FFFF0FFFh will return following values:
```
  ARM11 Opcode reads --> E59FF018h    (opcode LDR PC,[$+20h]
  ARM11 Data reads   --> [10140424h]  (CFG11_BOOTROM_OVERLAY_VAL)
```
This is intended for redirecting the CPU2/3 reset vectors, however, it does also affect all other exception vectors, including for CPU0/1 (so one should usually disable IRQs while having the overlay enabled).
See CFG11_MPCORE_BOOTCNT_CPU2/3 for details on starting CPU2/3.

10140424h - New3DS - CFG11_BOOTROM_OVERLAY_VAL (R/W)
```
  0-31  ARM11 Bootrom Overlay Data (aka CPU2/3 entrypoint)
```
Note: For redirecting to different entrypoints for CPU2/3, one can use opcode EE10nFB0h (mov Rn,p15,0,c0,c0,5) to obtain the current CPU number.

10140428h - New3DS - ??? (R/W)
```
  0     Unknown (R/W, only in New3DS mode)
  1-31  Unused (0)
```
Unknown.

10140FFCh - CFG11_SOCINFO (R)
```
  0     Always 1 (bootrom would do extra GPIO if it were cleared)      (R)
  1     Console Type        (0=Old3DS, 1=New3DS)                       (R)
  2     Max Clock on New3DS (0=536MHz, 1=804MHz) ;unused 0 on Old3DS   (R)
  3-15  Unused (0)
```
Unknown if there are any New3DS consoles released with bit2=0.
Bit2 is also related to CFG11_MPCORE_CNT bit8?

10141000h - CFG11_GPU_STATUS (R)
Unknown, reads as zero?

10141008h - CFG11_PTM_0 - PTM Enable/Trigger? (R/W)
```
  0     Unused (0) ?
  1     Unknown...?  (R/W)
  2     Unused (0) ?
  3     Unknown...?  (R/W)
  4     Unknown...?  (R/W)
  5     Unused (0) ?
  6     Unknown...?  (R/W)
  7     Unknown...?  (R/W)
  8     Unknown...?  (R/W)
  9-15  Unused (0) ?
  16-31 Unknown...? (0..FFFFh)  (R/W)
```
Unknown, bitmask FFFF01DAh is R/W.
Uh, what is a PTM... maybe "Play time PTM (rtc, pedometer, battery manager)"... which would be MCU related? There is also something called PTMSYSM (which seems to be also MCU related, but perhaps more related to Power/mode switching than Playing).
PTM might also refer to ARM's Program Trace Macrocell?
Setting bit3 or bit29 causes corresponding bits in CFG11_PTM_1 to get set, and that causes attempts to switch to New3DS mode (via CFG11_MPCORE_CLKCNT) to hang?

1014100Ch - CFG11_PTM_1 - PTM Status? (R/ack)

```
0-2   Unused (0) ?
3     Unknown, set if CFG11_PTM_0.bit3 is/was set  (write 1 to clear) (R/ack)
4-28  Unused (0) ?
29    Unknown, set if CFG11_PTM_0.bit29 is/was set (write 1 to clear) (R/ack)
30-31 Unused (0) ?
```

Unknown, initially zero, but bit3/29 can get set via CFG11_PTM_0.
Writing 1 seems to clear the bits (unless they are still set in CFG11_PTM_0).

10141100h - CFG11_TWLMODE_BOOT (R/W)

```
0-1   Setting from ARM7_CNT (Port 10018000h) (0=3DS, 1=NDS/DSi, 2=GBA)    (R)
2-14  Unused (0) ?
15    Enable GBA/NDS/DSi hardware   (can be set ONLY if bit0-1=nonzero) (R/W)
```

10141104h - CFG11_TWLMODE_1 (R/W)

```
0-14  Zero?
15    Unknown (R/W)
```

Observed 8000h when running under TWL_FIRM, 0 NATIVE_FIRM.

10141108h - CFG11_TWLMODE_2 (R?)

```
0-15  Unknown, reportedly "Bitfield", but looks like readonly, returns zero?
```

1014110Ah - CFG11_TWLMODE_HID_IRQ (R)

```
0-9   Button IRQ Enable flags (0=Ignore, 1=Select)
10-13 Not used
14    IRQ Enable Flag (0=Disable, 1=Enable)
15    IRQ Condition   (0=Logical OR, 1=Logical AND)
```

This is a readonly mirror of the ARM7 Key Interrupt Control register (KEYCNT, GBA/NDS Port 4000132h). The ARM7 uses this as wakeup condition for sleep mode, the ARM11 could forward that value to HID_PAD_IRQ when entering sleep mode, too. In homebrew code, one can also mis-use the register to send data from ARM7 to ARM11.
Note: The NDS does also have a separate KEYCNT register on ARM9, but that isn't forwarded to ARM11 side.

1014110Ch - CFG11_WIFIUNK (always 0, not R/W?)

```
0-3   unknown/unspecified
4     Wifi-related? Set to 1 very early in NWM-module... but isn't R/W... ?
5-xx  unknown/unspecified
```

Read-only or write-only, always returns zero?

10141110h - CFG11_TWLMODE_HID_MODE - R/W:0fffh

```
0-11  GBA/NDS Button Mode  (0=Auto/HID_PAD, 1=Manual/CFG11_TWLMODE_HID_SET)
12    Unknown, if any (not R/W, but reportedly used???) (dev hw maybe?)
13-15 Unused (0)
```

Set bits will use the corresponding values from CFG11_TWLMODE_HID_SET instead of allowing the hardware to read it from HID_PAD.
This is set to 0x1FFF (all buttons and the debug key) and CFG11_TWLMODE_HID_SET is set to 0 when the "Close this software and return to HOME Menu?" dialog is shown to prevent the button presses from propagating to the DS/GBA CPU.

10141112h - CFG11_TWLMODE_HID_SET - R/W:0fffh

```
0-11  Buttons   ... unknown, reportedly 0 when NOT pressed ??????????
12    Unknown, if any (not R/W, but reportedly used???) (dev hw maybe?)
13-15 Unused (0)
```

Works the same way as HID_PAD, but the values set here are only replaced in the HID_PAD seen by the NDS/GBA CPUs when the corresponding bits in CFG11_TWLMODE_HID_MODE are set.

10141114h - 2 - R/W:0287h CFG11_CODEC_0

10141116h - 2 - R/W:0387h CFG11_CODEC_1
10141118h - 1 - R/W:01h ?
10141119h - 1 - R/W:01h ?
10141120h - 1 - R:0 ?
Unknown.

10141200h - CFG11_GPU_CNT (R/W)
```
  0     Enable GPU registers at 10400000h and up (0=DANGER) (vram=snow?)
         "When this is unset VRAM is not accessible and triggers exceptions."
  1     Enable GPU_MEMFILL 0/1                   (0=Disable, 1=Enable)  (R/W)
  2     Enable reading GPU Internal registers?   (0=Hangs, 1=Enable)    (R/W)
  3     Enable reading GPU Internal registers?   (0=Hangs, 1=Enable)    (R/W)
  4     Enable GPU_MEMCOPY                       (0=Disable, 1=Enable)  (R/W)
  5     unknown, DANGER, hangs when cleared?     (0=DANGER, 1=Enable)
  6     Enable LCD Pixel Output                  (0=Off/fade, 1=Enable) (R/W)
  7-15  Unused (0)
  16    Enable LCD Backlight                     (0=Off/dark, 1=Enable) (R/W)
  17-31 Unused (0)
```

10141204h - CFG11_GPU_CNT2 (W)
```
  0     Power on GPU? (W)
  1-31  unknown/unspecified (0)
```

10141208h - CFG11_GPU_.....? (W)
```
  0     unknown/unspecified (used by bootrom) (W)
  1-7   unknown/unspecified (0)
```

10141210h - CFG11_GPU_FCRAM_CNT ;uh, that is... GPU and/or FCRAM related?
```
  0     Unknown...?                  (1=Enable?)                      (R/W)
  1     Enable/disable FCRAM         (1=Enable?)                      (R/W)
  2     Enable/disable operation in progress, uh? (busy/ready flag?)  (R?)
  3-xx  Unused (0)
```
Usually 07h.
Bit2 depends on what value was written to bit0-1?
```
  Write 00h --> read 04h                    ;\ARM11 hangs upon FCRAM access
  Write 01h --> read 01h (with bit2=0 here!) ; (GBA still works as bad as ever)
                         (only if OLD was 7?);
  Write 02h --> read 06h                    ;/
  Write 03h --> read 07h                    ;-ARM11 can access FCRAM
```
Bit2 doesn't seem to change after some time, ie. doesn't look like a progress/busy bit at all. Or maybe it is a ready bit, but hangs sometimes?
Unknown if this register is really GPU related.

10141220h - CFG11_CODEC_CNT
```
  0     Unknown...?            (R/W)
  1     turn on/off DSP ... ?  (R/W)
  2-xx  Unused (0)
```
This (what?) is the only time the ARM11 CODEC module uses any 10141xxxh registers. In one case CODEC module clears bit1 in register 10141114h, in the other case CODEC module sets bit1 in registers 10141114h and 10141116h.

10141224h - CFG11_CAMERA_CNT
```
  0     Camera Enable (needed for camera I2C bus)  (0=Disable, 1=Enable)
 (1)    Unused (0)      ;<--- reportedly "turn on/off cameras" uh???
  1-31  Unused (0)
```

10141230h - CFG11_DSP_CNT (R/W)
```
  0-1   Enable DSP registers at 10203000h (0,1,2=Disable, 3=Enable) (R/W)
  2-7   Unused (0)
```

Unknown what bit0-1 do exactly, maybe one is reset, and one is stop (they must be BOTH set though, otherwise registers at 10203000h are always zero).

10141240h - New3DS - unknown?

```
0     Unknown (R/W)
1-31  Unused (0) (?)
```

10141300h - New3DS - CFG11_MPCORE_CLKCNT

```
0-2    Desired Mode (0,1,2,3,4,5,6,7) (see below)                    (R/W)
3-14   Unused (0)
15     Mode Change IRQ Flag (0=None, 1=IRQ 58h) (write 1 to reset)   (R/ack)
16-18  Current Mode (0,1,3,5,7) (never 2,4,6) (see below)            (R)
19-31  Unused (0)
```

Mode values:

```
Mode 0: Old3DS Mode, 268MHz (1x), 128MB FCRAM, no L2C cache
Mode 1: New3DS Mode, 268MHz (1x), 256MB FCRAM, and L2C cache controller
Mode 3: New3DS Mode, 536MHz (2x), 256MB FCRAM, and L2C cache controller
Mode 5: New3DS Mode, 804MHz (3x), 256MB FCRAM, and L2C cache controller
Mode 2/4/6 are same as Mode 0. Mode 7 is same as Mode 5
```

Mode changes are applied ONLY once when all ARM11 cores are in Wait for IRQ state (eg. "WFI" opcode). At that point, the IRQ flag in bit15 is set, and IRQ 58h is triggered.

Mode changes (and IRQs) occur ONLY when changing to different modes:

```
- Changing between mode 0/1/3/5/7 (because they are different)
- Changing between mode 5/7 (although they appear to be same)
```

Mode changes (and IRQs) do NOT occur when:

```
- Rewriting the current mode value (because mode is already same)
- Changing between mode 0/2/4/6 (because they are all same as mode 0)
```

New3DS mode enables some registers (eg. 10141240h, which are always zero in Old3DS mode), and unlocks some Memory and I/O regions (which trigger data abort in Old3DS mode):

```
10130000h (L2B_0, L2B_1, Y2R_1, and MVD registers)
10206000h (newer CDMA controller)
17E10000h (L2C cache controller)
28000000h (Extra FCRAM)
```

Other New3DS memory isn't automatically enabled (and keeps triggering data abort even in New3DS mode, until explicitely enabling it):

```
08100000h (Extended ARM9 RAM, needs enable via CFG9_EXTMEMCNT9.bit0)
1F000000h (New VRAM aka QTM, needs enable via CFG11_MPCORE_CNT.bit0)
```

Clock changes affect the ARM11 CPU clock, including the MPCore Timer/Watchdog registers at 17E00600h (if desired, one can change their prescalers to maintain them running at same speed despite of the higher CPU clock).

The 804MHz mode should be used only if CFG11_SOCINFO.bit2 indicates that it is supported.

The IRQ flag in bit15 should be acknowledge before (or alongsides with) writing the desired clock value (needed if it was still set from an older un-acknowledged clock change).

IRQ 58h should be enabled to recover from WFI (or maybe anything like Hblank IRQ might work for that, too). CPSR.I flag does not need to be enabled, WFI does only wait for IRQs (but can recover without executing them). Using the old CP15 Wait-for-IRQ opcode instead of WFI does work, too.

10141304h - New3DS - CFG11_MPCORE_CNT

```
0     Enable New VRAM aka QTM at 1F000000h-1F3FFFFFh (0=Off, 1=On)   (R/W)
1-7   Unused (0)
8     Unknown... should be set only if CFG11_SOCINFO.bit2=1          (R/W)
9-xx  Unused (0)
```

Kernel11 sets this to 101h when bit2 in CFG11_SOCINFO is set otherwise 001h.

10141310h - New3DS - CFG11_MPCORE_BOOTCNT_CPU0 (R)
10141311h - New3DS - CFG11_MPCORE_BOOTCNT_CPU1 (R)

```
0-7   Fixed (always 30h, even when executing WFI opcode)              (R)
```

These are readonly dummy registers, somewhat indicating that CPU0/1 do exist and that they were already started (of which, CPU0 is actually running code, and CPU1 is initially hanging in a bootrom waitloop; which waits for FIRM code to send software interrupt 01h via Port 17E01F00h, and then jumps to [1FFFFFDCh]).

10141312h - New3DS - CFG11_MPCORE_BOOTCNT_CPU2 (R/W)
10141313h - New3DS - CFG11_MPCORE_BOOTCNT_CPU3 (R/W)

```
  0     Start/Stop CPU core         (0=Stop upon WFI, 1=Start/Enable)     (R/W)
  1     Must be 1 when starting CPU (0=Hangs when setting bit0, 1=Works)  (R/W)
  2-3   Unused (0)
  4     CPU started                 (0=No, 1=Yes, bit0 is/was set)          (R)
  5     CPU not in WFI state        (0=Executes WFI opcode, 1=Normal)       (R)
  6-7   Unused (0)
```

Starting CPU2/3 must be done with CFG11_BOOTROM_OVERLAY_CNT/VAL (the default reset vector in ARM11 bootrom doesn't support booting CPU2/3).
Bit0 enables CPU2/3 (and alongsides enables corresponding registers at 17E00900h/17E00A00h).
Bit1 must be set alongsides with bit0 (otherwise the CPU hangs somehow), once when the CPU is started, toggling bit1 won't affect the CPU operation; clearing bit1 is commonly done to notify CPU0 that the CPU has started.
Note: When starting/stopping CPU2/3, one should also notify the hardware about the new state (via Port 17E00008h, SCU CPU Status Register).

3DS Config - AXI Registers

PrimeCell High-Performance AXI Bus Matrix (HPM) (PL301) Revision: r1p2
This is something for configuring interactions between Master Interfaces (MIs) and Slave Interfaces (SIs). The datasheet contains only meaningless blurb in a too-big-too-fail language. Guessing between the lines, Master does probably refer to CPUs and DMA controllers, and Slave might refer to Memory Chips.

```
  1020F000h-3FCh   Unused (0)                                              (-)
  1020F400h+MI*20h QoS Tidemark for Master MI=00h                         (R/W)
  1020F404h+MI*20h QoS Access Control for Master MI=00h                   (R/W)
  1020F408h+MI*20h AR Channel Arbitration value for MI=00h..NumMI-1       (R/W)
  1020F40Ch+MI*20h AW Channel Arbitration value for MI=00h..NumMI-1       (R/W)
  1020F800h-FBCh   Reserved (0)                                            (-)
  1020FFC0h        PrimeCell Configuration Register 0 NumSI's (07h/0Ah)    (R)
  1020FFC4h        PrimeCell Configuration Register 1 NumMI's (11h/16h)    (R)
  1020FFC8h        PrimeCell Configuration Register 2 Zero    (00h)        (R)
  1020FFCCh        PrimeCell Configuration Register 3 Zero    (00h)        (R)
  1020FFD0h-FDCh   Reserved (0)                                            (-)
  1020FFE0h-FECh   PrimeCell Peripheral Register 0,1,2,3 (01h,13h,x4h,00h) (R)
  1020FFF0h-FFCh   PrimeCell ID Register 0,1,2,3         (0Dh,F0h,05h,B1h) (R)
```

These registers allow to read some fixed settings, and to change a few variable settings (there must be a lot more fixed internal settings assigned at manufacturing time, but one cannot read or change them).
The 3DS seems to support 1020F400h/1020F404h for Master 0 only.
Whilst 1020F408h/1020F40Ch are supported for Master 0 and up.

1020F400h+MI*20h - QoS Tidemark for Master MI=00h (R/W)

```
  0-6   Max number of outstanding transactions before activating QoS (0..7Fh)
  7-31  Unused (0)
```

A value of 00h does completely disable QoS (instead of instantly triggering it upon 0 outstanding transactions).
QoS is short for Quality of Service (whatever than means).

1020F404h+MI*20h - QoS Access Control for Master MI=00h (R/W)

```
  0-6   Permit Slave 0-6 to use reserved slots (1=Yes) ;\Old3DS mode
  7-31  Unused (0)                                     ;/
  0-9   Permit Slave 0-9 to use reserved slots (1=Yes) ;\New3DS mode
  10-31 Unused (0)                                     ;/
```

1020F408h+MI*20h AR Channel Arbitration value for MI=00h..NumMI-1 (R/W)
1020F40Ch+MI*20h AW Channel Arbitration value for MI=00h..NumMI-1 (R/W)
Arbitration for AXI read (AR) and AXI write (AW) address channel signals.
The meaning of these register depends on the chip-configuration. There are three possible modes for each MI (of which, the 3DS uses LRG for MI=00h, and Fixed RR for MI=01h and up).

```
Programmable Least Recently Granted (LRG) arbitration (3DS: used for MI=00h)
 Write ii00pp00h   ;set priority for interface                        ;-write
 Write FF0000iih   ;select interface for reading                      ;\read
 Read  0000ppiih   ;read priority for previously selected interface   ;/
Fixed Round-robin (RR) arbitration scheme (3DS: used for MI=01h..NumMI-1)
 Write xxxxxxxxh   ;ignored (values are fixed)                        ;-write
 Write FF0000ssh   ;select slot for reading                           ;\read
 Read  000000iih   ;read interface for previously selected slot       ;/
Programmable Round-robin (RR) arbitration scheme (3DS: not used)
 Write ss0000iih   ;set interface for slot                            ;-write
 Write FF0000ssh   ;select slot for reading                           ;\read
 Read  000000iih   ;read interface for previously selected slot       ;/
```

Whereas, the parameter bits are:

```
 pp = Priority               (00h..FFh; 00h=Highest, FFh=Lowest)  ;for LRG
 ss = Slot number            (00h..unknown max value)             ;for RR
 ii = Slave interface number (00h..NumSI-1)
 FF = Fixed code for reading (FFh)
 00 = Unused/reserved        (00h)
```

The following settings exist on New3DS (the fixed values are same for AR+AW):

```
 Master  Old3DS Mode                        New3DS Mode
 MI=00h  pp pp pp pp pp pp pp -- -- -- --   pp pp pp pp pp pp pp pp pp pp --
 MI=01h  -- 01 02 03 04 05 06 -- -- -- --   -- 01 02 03 04 05 06 07 08 09 --
 MI=02h  01 02 05 06 -- -- -- -- -- -- --   01 02 05 06 08 09 -- -- -- -- --
 MI=03h  01 02 04 05 06 -- -- -- -- -- --   01 02 04 05 06 08 09 -- -- -- --
 MI=04h  01 02 04 -- -- -- -- -- -- -- --   01 02 04 08 -- -- -- -- -- -- --
 MI=05h  01 02 04 -- -- -- -- -- -- -- --   01 02 04 08 -- -- -- -- -- -- --
 MI=06h  01 02 04 -- -- -- -- -- -- -- --   01 02 04 08 -- -- -- -- -- -- --
 MI=07h  01 02 -- -- -- -- -- -- -- -- --   01 02 08 -- -- -- -- -- -- -- --
 MI=08h  01 02 -- -- -- -- -- -- -- -- --   01 02 08 -- -- -- -- -- -- -- --
 MI=09h  01 02 -- -- -- -- -- -- -- -- --   01 02 08 -- -- -- -- -- -- -- --
 MI=0Ah  01 02 05 -- -- -- -- -- -- -- --   01 02 05 08 -- -- -- -- -- -- --
 MI=0Bh  -- -- -- -- -- -- -- -- -- -- --   02 08 -- -- -- -- -- -- -- -- --
 MI=0Ch  -- -- -- -- -- -- -- -- -- -- --   08 02 -- -- -- -- -- -- -- -- --
 MI=0Dh  -- -- -- -- -- -- -- -- -- -- --   01 08 -- -- -- -- -- -- -- -- --
 MI=0Eh  -- -- -- -- -- -- -- -- -- -- --   -- -- -- -- -- -- -- -- -- -- --
 MI=0Fh  -- -- -- -- -- -- -- -- -- -- --   -- -- -- -- -- -- -- -- -- -- --
 MI=10h  -- -- -- -- -- -- -- -- -- -- --   -- -- -- -- -- -- -- -- -- -- --
 MI=11h  -- -- -- -- -- -- -- -- -- -- --   01 08 -- -- -- -- -- -- -- -- --
 MI=12h  -- -- -- -- -- -- -- -- -- -- --   01 08 -- -- -- -- -- -- -- -- --
 MI=13h  -- -- -- -- -- -- -- -- -- -- --   01 08 -- -- -- -- -- -- -- -- --
 MI=14h  -- -- -- -- -- -- -- -- -- -- --   -- 01 02 03 04 05 06 07 08 09 --
 MI=15h  -- -- -- -- -- -- -- -- -- -- --   -- 01 02 03 04 05 06 07 08 09 --
```

Entries with value "01..09" are fixed (and have same values for AR and AW).
Entries marked "--" are reading as zero (but there seems to be no way to distinguish between SI=00h and SI=None).
The priority values (pp) for MI=00h are ininitally 00h, but can be changed, New3DS has 34 priority values in total (7+7 for AR+AW in Old3DS mode, and another 10+10 for AR+AW in New3DS mode).

1020FFC0h PrimeCell Configuration Register 0 Num SIs (R)
1020FFC4h PrimeCell Configuration Register 1 Num MIs (R)

```
 0-7   Number of Master/Slave Interfaces (MIs/SIs) (01h..20h)
 8-31  -
```

The values here change depending on whether running in Old3DS or New3DS mode.

```
 Old3DS Mode: NumMI=11h, NumSI=07h
 New3DS Mode: NumMI=16h, NumSI=0Ah
```

1020FFC8h PrimeCell Configuration Register 2 Zero (R)
1020FFCCh PrimeCell Configuration Register 3 Zero (R)
```
0-31  Zero
```

1020FFE0h-FECh - PrimeCell Peripheral Register 0-3 (R)
This region contains four 32bit registers (01h,13h,x4h,00h). One is supposed to extract the lower 8bit of these 32bit values, and then to merge them into a "conceptual-32bit-value" (00x41301h). And then interprete it as so:
```
0-11  Part number           (301h=HPM)
12-19 Designer              (41h=ARM)
20-23 Revision              (1=r1p0, 2=r1p1, 3=r1p2)
24-31 Reserved (undef)
```
This value is fixed=00341301h on New3DS (ie. r1p2, no matter if running in Old3DS or New3DS mode).
The value on actual Old3DS is unknown (although, some consoles are reportedly using r1p0?).

1020FFF0h-FFCh - PrimeCell ID Register 0-3 (R)
This region contains four 32bit registers (0Dh,F0h,05h,B1h). One is supposed to extract the lower 8bit of these 32bit values, and then to merge them into a "conceptual-32bit-value" (B105F00Dh). And then interprete it as so:
```
0-31  Component ID (B105F00Dh) (same ID as for Corelink DMA controller)
```

Guesses on possible Master Interfaces (MIs)
Old3DS = 11h MI's
```
1xARM11 (with 2 CPU cores)
1xCDMA  (with 8 channels)
1xCSND  (with 32+2 sound+capture channels)
1xGPU   (internal rendering, external to 2 LCD's, and memcopy/memfill)
1xDSP
1xARM9                              ;\
1xXDMA  (with 4 channels)           ; ARM9
1xNDMA  (with 8 channels)           ;
1xDMA   (with 4 channels)           ;/
1xARM7                              ;\
1xNDMA  (with 4 channels)           ; ARM7
1xDMA   (with 4 channels)           ; (can't really share ARM11 bus though)
1xNDS/GBA GPU (2x 2D and 1x 3D)     ;
1xNDS/GBA Sound (with 15+2 channels)  ;/
```
New3DS = 16h MIs (five more than Old3DS)
```
0xNewARM11 (but with 2 more CPU cores)
1xNewCDMA  (with 8 channels)
1xMVD
1xLevel 2 Cache Controller
```

Guesses on possible Slave Interfaces (SIs)
Old3DS = 07h SI's
```
FCRAM
VRAM
ARM9 RAM
DSP RAM
AXI RAM
BIOS ROM(s)
I/O Area(s)
```
New3DS = 0Ah SIs (three more than Old3DS)
```
Extended VRAM? aka QTM?
Extended ARM9 RAM
Extended FCRAM
```

Datasheets
Reportedly "CoreLink NIC-301 r1p0" (aka NIC-301 Network Interconnect)
```
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0422a/CHDGHIID.html
DDI0422.pdf
```

But, the ID values don't match up (at least not on New3DS, it's actually: r1p2)
DDI0422D_hpm_pl301_r1p2_ts.pdf

3DS Config - L2C-310 Level 2 Cache Controller (New3DS)

L2C-310 r3p3, Level 2 Cache Controller (New3DS only)

```
17E10000h     L2C_CACHE_ID             R  410000C9h ;\Cache ID and Cache Type
17E10004h     L2C_CACHE_TYPE           R  9E440440h ;/
17E10100h     L2C_CONTROL              RW 00000000h ;\
17E10104h     L2C_AUX_CONTROL          RW 02090000h ; Control
17E10108h     L2C_TAG_RAM_CONTROL      RW 00000111h ;
17E1010Ch     L2C_DATA_RAM_CONTROL     RW 00000221h ;/
17E10200h     L2C_EV_COUNTER_CTRL      RW 00000000h ;\
17E10204h     L2C_EV_COUNTER1_CFG      RW 00000000h ; Interrupt and
17E10208h     L2C_EV_COUNTER0_CFG      RW 00000000h ; Counter Control
17E1020Ch     L2C_EV_COUNTER1          RW 00000000h ;
17E10210h     L2C_EV_COUNTER0          RW 00000000h ;
17E10214h     L2C_INT_MASK             RW 00000000h ;  ;\
17E10218h     L2C_INT_STATUS_MASKED    R  00000000h ;  ; Interrupts
17E1021Ch     L2C_INT_STATUS_RAW       R  00000000h ;  ;
17E10220h     L2C_INT_CLEAR            W  00000000h ;/ ;/
17E10730h     L2C_CACHE_SYNC           RW 00000000h ;\
17E10770h     L2C_INV_PA               RW 00000000h ;
17E1077Ch     L2C_INV_WAY              RW 00000000h ; Cache Maintenance
17E107B0h     L2C_CLEAN_PA             RW 00000000h ; Operations
17E107B8h     L2C_CLEAN_INDEX          RW 00000000h ;
17E107BCh     L2C_CLEAN_WAY            RW 00000000h ;
17E107F0h     L2C_CLEAN_INV_PA         RW 00000000h ;
17E107F8h     L2C_CLEAN_INV_INDEX      RW 00000000h ;
17E107FCh     L2C_CLEAN_INV_WAY        RW 00000000h ;/
17E10900h+N*8 L2C_D_LOCKDOWN_0..7      RW 00000000h ;\
17E10904h+N*8 L2C_I_LOCKDOWN_0..7      RW 00000000h ; Cache Lockdown
17E10950h     L2C_LOCK_LINE_EN         RW 00000000h ;
17E10954h     L2C_UNLOCK_WAY           RW 00000000h ;/
17E10C00h     L2C_ADDR_FILTERING_START RW 00000000h ;\Address Filtering
17E10C04h     L2C_ADDR_FILTERING_END   RW 00000000h ;/
17E10F40h     L2C_DEBUG_CTRL           RW 00000004h ;\
17E10F60h     L2C_PREFETCH_CTRL        RW 04000000h ; Debug, Prefetch, Power
17E10F80h     L2C_POWER_CTRL           RW 00000000h ;/
```

Caution: L2C registers can be read via LDR only (LDRB/LDRH cause data abort).
Note: Some L2C registers are write-able only with Secure access (S), not with Non-secure access (NS), but that seems to apply only for TrustZone extension (which doesn't seem to exist in 3DS).
Official specs: DDI0246H_l2c310_r3p3_trm.pdf

_____ Cache ID and Cache Type (NS and S) _____

17E10000h - L2C_CACHE_ID - Cache ID Register (410000C9h) (R)

```
0-5   RTL release              (9=r3p3)
6-9   Part number              (3=L2C-310)
10-15 CACHEID pins             (reads as 0 on New3DS)
16-23 Reserved (0)
24-31 Implementer              (41h=ARM)
```

17E10004h - L2C_CACHE_TYPE - Cache Type Register (9E440440h) (R)

```
0-1   L2 cache line length     (0=32 bytes)                    ;\
2-5   Reserved (0)                                             ;
6     L2 associativity         (from L2C_AUX_CONTROL.bit16)     ; instruction
7     Reserved (0)                                             ;
8-10  Isize L2 cache way size  (from L2C_AUX_CONTROL.bit19-17) ;
11    Reserved (0)                                             ;/
```

```
  12-13 L2 cache line length      (0=32 bytes)                    ;\
  14-17 Reserved (0)                                              ;
  18    L2 associativity          (from L2C_AUX_CONTROL.bit16)     ; data
  19    Reserved (0)                                              ;
  20-22 Dsize L2 cache way size   (from L2C_AUX_CONTROL.bit19-17) ;
  23    Reserved (0)                                              ;/
  24    Harvard                   (0=Unified, 1=Harvard)          ;-harvard
  25    Lockdown by Line option   (0=Off, 1=On)                   ;\
  26    Lockdown by Master option (0=Off, 1=On)                   ; ctype
  27-28 Fixed (always 3)                                          ;/
  29-30 Reserved (0)
  31    Data banking              (0=Not implemented, 1=Implemented)

_____________________ Control (Write S, Read NS and S) _____________________
```

17E10100h - L2C_CONTROL - Control Register (R/W)

```
  0     L2 Cache enable           (0=Disable, 1=Enable)
  1-31  Reserved (0)
```

Caution: The cache seems to contain garbage on power-up. Set L2C_INV_WAY=FFFFh, then wait for L2C_INV_WAY=0 before enabling L2C_CONTROL.

17E10104h - L2C_AUX_CONTROL - Auxiliary Control Register (02090000h) (R/W)

```
  0     Full Line of Write Zero Enable (0=Disable, 1=Enable)
  1-7   Reserved (0) (actually R/W, but unknown/undocumented)
  8     Reserved (0) (always 0)
  9     Reserved (0) (actually R/W, but unknown/undocumented)
  10    Priority for Strongly Ordered and Device Reads Enable (0=Low, 1=High)
  11    Limit  (0=Device writes can use all slots, 1=Ensure one Memory slot)
  12    Exclusive cache configuration      (0=Disable, 1=Enable)
  13    Shared Attribute Invalidate Enable (0=Disable, 1=Enable if no override)
  14-15 Reserved (0) (actually R/W, but unknown/undocumented)
  16    Associativity              (0=8-way, 1=16-way)
  17-19 Way-size (1=16K, 2=32K, 3=64K, 4=128K, 5=256K, 6=512K, 0/7=Same as 1/6)
  20    Event monitor bus enable   (0=Disable, 1=Enable)
  21    Parity enable              (0=Disable, 1=Enable)
  22    Shared attribute override  (0=No, 1=Ignore Shared Attrubute)
  23-24 Force write allocate (0=Use WA, 1=ForceWA=0, 2=ForceWA=1, 3=Same as 0?)
  25    Cache Replacement Policy   (0=Pseudo-random/LFSR, 1=Round-robin)
  26    Lockdown Register Writes   (0=Secure only, 1=Allow non-secure)
  27    Interrupt MASK/CLEAR Access (0=Secure only, 1=Allow non-secure)
  28    Data Prefetch Enable       (0=Disable, 1=Enable)
  29    Instruction Prefetch Enable (0=Disable, 1=Enable)
  30    Early BRESP Enable         (0=Disable, 1=Enable, Early write response)
  31    Reserved (0) (actually R/W, but unknown/undocumented)
```

Cache size is reportedly "2MB" on New3DS, which is probably meant to be 2Mbyte, ie. 16 ways of 128Kbyte each, and that shared for both code and data caching?

17E10108h - L2C_TAG_RAM_CONTROL - Tag RAM Latency Control (00000111h) (R/W)
17E1010Ch - L2C_DATA_RAM_CONTROL - Data RAM Latency Control (00000221h) (R/W)

```
  0-2   RAM setup latency          (0-7 = 1..8 cycles of latency)
  3     Reserved (0)
  4-6   RAM read access latency    (0-7 = 1..8 cycles of latency)
  7     Reserved (0)
  8-10  RAM write access latency   (0-7 = 1..8 cycles of latency)
  11-31 Reserved (0)
```

Uh, is that the latency of the Cache Memory (ie. not the external "cached" memory)?

```
_________________ Interrupt and Counter Control (NS and S) _________________
```

17E10200h - L2C_EV_COUNTER_CTRL - Event Counter Control (R/W)

```
  0     Event Counting Enable      (0=Disable, 1=Enable)    (R/W)
  1     Event Counter 0 Reset      (0=No change, 1=Reset)   (W)
```

```
  2     Event Counter 1 Reset       (0=No change, 1=Reset)   (W)
  3-31  Reserved (0)
```

17E10204h - L2C_EV_COUNTER1_CFG - Event Counter 1 Configuration (R/W)
17E10208h - L2C_EV_COUNTER0_CFG - Event Counter 0 Configuration (R/W)

```
  0-1   Event counter interrupt generation  (00h-03h, see below)
  2-5   Counter event source                (00h-0Fh, see below)
  6-31  Reserved (0)
```

Event counter interrupt generation:

```
  00h Disabled                          ;count, without irq
  01h Enabled: Increment condition      ;count, with irq on any increment
  02h Enabled: Overflow condition       ;count, with irq on overflow
  03h Interrupt generation is disabled  ;count, without irq (same as 0?)
```

Counter event source:

```
  00h  -        Counter Disabled
  01h  CO       Eviction, CastOUT, of a line from the L2 cache
  02h  DRHIT    Data read hit in the L2 cache
  03h  DRREQ    Data read lookup to the L2 cache
  04h  DWHIT    Data write hit in the L2 cache
  05h  DWREQ    Data write lookup to the L2 cache
  06h  DWTREQ   Data write lookup to the L2 cache with Write-Through attribute
  07h  IRHIT    Instruction read hit in the L2 cache
  08h  IRREQ    Instruction read lookup to the L2 cache
  09h  WA       Allocation into the L2 cache caused by a write, with
                  Write-Allocate attribute, miss
  0Ah  IPFALLOC Allocation of a prefetch generated by L2C-310 into the L2 cache
  0Bh  EPFHIT   Prefetch hint hits in the L2 cache
  0Ch  EPFALLOC Prefetch hint allocated into the L2 cache
  0Dh  SRRCVD   Speculative read received by slave port(s)
  0Eh  SRCONF   Speculative read confirmed in slave port(s)
  0Fh  EPFRCVD  Prefetch hint received by slave port(s)
  Note: All REQ lookups will subsequently result in a hit or miss.
```

17E1020Ch - L2C_EV_COUNTER1 - Event counter 1 value (R/W)
17E10210h - L2C_EV_COUNTER0 - Event counter 0 value (R/W)

```
  0-31  Counter value, incremented on selected event
```

If a counter reaches its maximum value, it saturates at that value until it is reset.

17E10214h - L2C_INT_MASK - Interrupt Mask (0=Disable, 1=Enable) (R/W)
17E10218h - L2C_INT_STATUS_MASKED - Masked Interrupt Status Register (R)
17E1021Ch - L2C_INT_STATUS_RAW - Raw Interrupt Status (1=IRQ) (R)
17E10220h - L2C_INT_CLEAR - Interrupt Clear (0=No change, 1=Clear) (W)

```
  0     ECNTR: Event Counter 0 and 1 Overflow/Increment
  1     PARRT: Parity Error on L2 tag RAM, Read
  2     PARRD: Parity Error on L2 data RAM, Read
  3     ERRWT: Error on L2 tag RAM, Write
  4     ERRWD: Error on L2 data RAM, Write
  5     ERRRT: Error on L2 tag RAM, Read
  6     ERRRD: Error on L2 data RAM, Read
  7     SLVERR: SLVERR from L3
  8     DECERR: DECERR from L3
  9-31  Reserved (0)
```

Note: STATUS_MASKED is same as STATUS_RAW, but ANDed with MASK.
The IRQ triggers interrupt 76h.

Cache Maintenance Operations

17E10730h - L2C_CACHE_SYNC - Cache Maintenance Operations (R and W)

```
  0     C, When writing: Must be 0 (trigger cache sync...?)
  0     C, When reading: Background/Way operation is in progress (0=No, 1=Yes)
  1-31  Reserved (0)
```

17E10770h - L2C_INV_PA - Invalidate by Physical Address (R/W)
17E107B0h - L2C_CLEAN_PA - Clean by Physical Address (R/W)
17E107F0h - L2C_CLEAN_INV_PA - Clean+Invalidate by Physical Address (R/W)

```
0     C (uh, is that same meaning as in L2C_CACHE_SYNC.bit0?)   (R?)
1-4   Reserved (0)
5-xx  Index                                                    (W?)
xx-31 Tag                                                      (W?)
```

17E1077Ch - L2C_INV_WAY - Invalidate by Way (R/W)
17E107BCh - L2C_CLEAN_WAY - Clean by Way (R/W)
17E107FCh - L2C_CLEAN_INV_WAY - Clean+Invalidate by Way (R/W)

```
0-15  Way bits (for way 0..15) (1=Trigger/busy?)               (R/W)
16-31 Reserved (0)
```

17E107B8h - L2C_CLEAN_INDEX - Clean by Index (R/W)
17E107F8h - L2C_CLEAN_INV_INDEX - Clean+Invalidate by Index (R/W)

```
0     C (uh, is that same meaning as in L2C_CACHE_SYNC.bit0?)   (R?)
1-4   Reserved (0)
5-xx  Index                                                    (W?)
xx-xx Reserved (0)
28-31 Way number (0..15)                                       (W?)
```

_____ Cache Lockdown _____

17E10900h+N*8 - L2C_D_LOCKDOWN_0..7 - Data Cache lockdown 0-7 (R/W)
17E10904h+N*8 - L2C_I_LOCKDOWN_0..7 - Instruction Cache lockdown 0-7 (R/W)

```
0-15  DATALOCK/INSTRLOCK 000..111  (use when AyUSERSx[7:5]=000b..111b) ;way?
16-31 Reserved (0)
```

17E10950h - L2C_LOCK_LINE_EN - Lockdown by Line Enable (R/W)

```
0     Lockdown by Line Enable (0=Disable, 1=Enable)
1-31  Reserved (0)
```

17E10954h - L2C_UNLOCK_WAY - Unlock all Lines by Way (R/W)

```
0-15  Unlock all Lines by Way operation (0=No/off, 1=Unlock/Busy) ;way0-15
16-31 Reserved (0)
```

Caution: When busy, all other L2C registers are becoming readonly (so after writing, one MUST wait until above busy flags go off).

_____ Address Filtering (Write S, Read NS and S) _____

17E10C00h - L2C_ADDR_FILTERING_START - Address filtering Start (R/W)
17E10C04h - L2C_ADDR_FILTERING_END - Address filtering End (R/W)

Not implemented, always zero in New3DS. Allows to redirect a whole address range to master 1 (when two masters are implemented).

```
0     Address Filtering Enable (0=Disable, 1=Enable)  ;<-- in Start register
0     Reserved (0)                                    ;<-- in End register
1-19  Reserved (0)
20-31 Address Filtering Start/End Address bit31-20
```

_____ Debug, Prefetch and Power (Write S, Read NS and S) _____

17E10F40h - L2C_DEBUG_CTRL - Debug Register (00000004h) (R/W)

```
0     Disable cache linefill (0=Enable, 1=Disable cache linefills)      (R/W)
1     Disable write-back     (0=Write-back, 1=Force Write-through)      (R/W)
2     Secure Privileged Non-Invasive Debug Enable SPNIDEN option (1=on?)  (R)
3-31  Reserved (0)
```

17E10F60h - L2C_PREFETCH_CTRL - Prefetch Control Register (04000000h) (R/W)

```
0-4   Prefetch Offset (must be 0-7, 15, 23, or 31) (other=Unsupported)
5-20  Reserved (0) (always 0)
21    Not same AXI ID on exclusive sequence enable   (0=Same ID, 1=Not same)
22    Reserved (0) (actually R/W, but unknown/undocumented)
23    Incr Double Linefill enable (allow 8x64bit)    (0=Disable, 1=Allow)
24    Prefetch drop, Discard prefetch reads to L3    (0=Disable, 1=Enable)
25    Reserved (0) (actually R/W, but unknown/undocumented)
26    Speculative Read Synthesis Option (read-only)  (0=On, 1=Off)        (R)
27    Double linefill on WRAP read disable           (0=Enable, 1=Disable)
28    Data prefetch enable                           (0=Disable, 1=Enable)
29    Instruction prefetch enable                    (0=Disable, 1=Enable)
30    Double Linefill, Read bursts to L3 on L2 miss  (0=4x64bit, 1=8x64bit)
31    Reserved (0) (actually R/W, but unknown/undocumented)
```

17E10F80h - L2C_POWER_CTRL - Power Control Register (R/W)

```
0     Standby mode enable         (0=Disable, 1=Enable)
1     Dynamic clock gating enable (0=Disable, 1=Enable)
2     Reserved (0) (actually R/W, but unknown/undocumented)
3-31  Reserved (0) (always 0)
```

3DS Config - ARM7 Registers (GBA/NDS/DSi Mode)

The somewhat misnamed "ARM7" registers are actually ARM9 registers, used to configure GBA/NDS/DSi mode.

```
10018000h 1   ARM7_CNT               ;\GBA/NDS/DSi mode
10018080h 20h ARM7_BOOTCODE          ;/
10018100h 2   ARM7_SAVE_TYPE         ;\GBA savedata
10018104h 2   ARM7_SAVE_CNT          ;/
10018108h 2   ARM7_RTC_CNT           ;\
10018110h 4   ARM7_RTC_BCD_DATE      ;
10018114h 4   ARM7_RTC_BCD_TIME      ; GBA real time clock
10018118h 4   ARM7_RTC_HEX_TIME      ;
1001811Ch 4   ARM7_RTC_HEX_DATE      ;/
10018120h 4   ARM7_SAVE_CFG_?        ;\
10018124h 4   ARM7_SAVE_CFG_?        ; GBA savedata timings?
10018128h 4   ARM7_SAVE_CFG_?        ;
1001812Ch 4   ARM7_SAVE_CFG_?        ;/
04700000h 4   <-- on ARM7 side, disable bootcode overlay
```

For specs on GBA/NDS/DSi mode:
GBA Reference
NDS Reference
DSi Reference
The ARM7_RTC_xxx registers are emulating the GBA cartridge RTC. The RTC for NDS/DSi mode is emulated separately via registers in the 3DS GPIO register space:
3DS GPIO Registers
The GBA/NDS/DSi video must be forwarded to 3DS framebuffers via ARM11 DMA:
3DS Video LGY Registers (Legacy GBA/NDS Video to Framebuffer)
GBA Audio requires enabling amplifiers via TSC register, and enabling GBA sound via CODEC_SNDEXCNT. NDS/DSi sound probably requires something similar, and more extensive TSC initialization for NDS or DSi touchscreen mode; and adjusting the TSC calibration points depending on whether using LGYFB scaling.

10018000h - ARM7_CNT (R/W)

```
0-1   Console Mode (0=3DS, 1=NDS/DSi, 2=GBA, 3=Auto-replaced by 0)      (R/W)
2-31  Unused (0)
```

After writing ARM7_CNT, apply the mode value by writing [10141100h].bit15=1 (CFG11_TWLMODE_BOOT). GBA/NDS/DSi mode will start ARM7, executing the ARM7_BOOTCODE.

NDS/DSi mode changes the ARM9 memory map (to survive the changed memory map, ARM9 should execute ITCM code during the mode change). GBA mode does keep ARM9 in 3DS mode. ARM11 is always kept running in 3DS mode.

10018080h..1001809Fh - ARM7_BOOTCODE (32 bytes) (R/W)
These 32 bytes do overlay the exception vectors at 00000000h..0000001Fh in the ARM7 BIOS ROMs (for GBA/NDS/DSi mode). The ARM7 reset vector opcode(s) are executed after switching to GBA/NDS/DSi mode (see ARM7_CNT CFG11_TWLMODE_BOOT.bit15).
After booting, ARM7 can disable the BOOTCODE overlay by writing [4700000h]=1.

ARM7:04700000h - on ARM7 side, disable bootcode overlay (W)
```
0     Disable ARM7_BOOTCODE overlay (0=No, 1=Disable) (SET-ONCE)
1-31  Unused (0)
```

GBA Mode Cartridge Savedata and RTC

10018100h - ARM7_SAVE_TYPE (R/W)
```
0-3   GBA Cartridge Type (00h-0Fh, see below)                        (R/W)
4-15  Unused (0)
```
Type values (same as used in the footer of the GBA ROM-image):
```
00h = ROM 16.0Mbyte, EEPROM 0.5Kbyte (in upper 16Mbyte of ROM area)
01h = ROM 31.9Mbyte, EEPROM 0.5Kbyte (in upper 100h byte of ROM area)
02h = ROM 16.0Mbyte, EEPROM 8Kbyte   (in upper 16Mbyte of ROM area)
03h = ROM 31.9Mbyte, EEPROM 8Kbyte   (in upper 100h byte of ROM area)
04h = ROM 32Mbyte, FLASH 64Kbyte, RTC   ;\(FLASH ID=3D1Fh, Atmel)
05h = ROM 32Mbyte, FLASH 64Kbyte        ;/
06h = ROM 32Mbyte, FLASH 64Kbyte, RTC   ;\(FLASH ID=D4BFh, SST)
07h = ROM 32Mbyte, FLASH 64Kbyte        ;/
08h = ROM 32Mbyte, FLASH 64Kbyte, RTC   ;\(FLASH ID=1B32h, Panasonic)
09h = ROM 32Mbyte, FLASH 64Kbyte        ;/
0Ah = ROM 32Mbyte, FLASH 128Kbyte, RTC  ;\(FLASH ID=09C2h, Macronix)
0Bh = ROM 32Mbyte, FLASH 128Kbyte       ;/
0Ch = ROM 32Mbyte, FLASH 128Kbyte, RTC  ;\(FLASH ID=1362h, Sanyo)
0Dh = ROM 32Mbyte, FLASH 128Kbyte       ;/
0Eh = ROM 32Mbyte, SRAM 32Kbyte         ;-SRAM
0Fh = ROM 32Mbyte                       ;-Raw ROM
```

10018104h - ARM7_SAVE_CNT (R/W)
```
0     Savedata mapping (0=GBA:0E000000h, 1=3DS:08080000h)            (R/W)
1-15  Unused (0)
```

10018108h - ARM7_RTC_CNT (W and R)
```
0     Write   (0=No change, 1=Apply RTC_HEX)                          (W)
1     Read    (0=No change, 1=Latch RTC_BCD and RTC_HEX)              (W)
2-13  Unused (0)
14    Write Error Flag      (0=Okay, 1=Error, invalid data)           (R)
15    Write/Read Busy Flag  (0=Ready, 1=Busy)                         (R)
```
To get the current time:
```
Set Read flag, wait until busy=0, then read RTC_HEX or RTC_BCD registers
```
To set the current time:
```
Write RTC_HEX registers, then set Write flag, then wait until busy=0
(this does also set BCD registers, the hardware auto-converts HEX to BCD)
```
The initial time on power-up in 01 Jan 2000, 00:00:00. The actual battery backed time can be obtained from MCU[30h..36h].
The GBA software can access the RTC via port 080000C4h, 080000C6h, 080000C8h.

10018110h - ARM7_RTC_BCD_DATE (R)
```
0-7   Year  BCD   (00h..99h)
8-15  Month BCD   (01h..12h)
```

```
16-23 Day   BCD   (01h..31h)
24-31 Day of Week (00h..06h) (WHAT=Monday?)
```

10018114h - ARM7_RTC_BCD_TIME (R)
```
0-7   Hour    BCD (00h..23h) (always 24-hours, even in AM/PM mode)
8-15  Minute  BCD (00h..59h)
16-23 Second  BCD (00h..59h)
24-31 Zero        (00h)
```

10018118h - ARM7_RTC_HEX_TIME (R=Latched Read value, W=Written value)
```
0-6   Second      (signed -40h..+3Fh, usually 00h..3Bh)
7     Force Reset (0=Normal, 1=Force 1st Jan 2000, 00:00:00)
8-14  Minute      (signed -40h..+3Fh, usually 00h..3Bh)
15    12/24-hour  (0=12 hour, 1=24 hour) (for GBA side, both with AM/PM flag)
16-21 Hour        (signed -20h..+1Fh, usually 00h..17h)
22-23 Unused      (0)
24-27 Day of Week (signed -08h..+07h, usually 00h..06h) (WHAT=Monday?)
28-30 Unknown     (can be 0..7)
31    Error       (0=Normal, 1=Triggers error)
```

1001811Ch - ARM7_RTC_HEX_DATE (R=Latched Read value, W=Written value)
```
0-15  Days since 1st Jan 2000 (0000h..8EACh=100 years, Bigger=Triggers error)
16-31 Unknown                 (can be 0000h..FFFFh)
```

10018120h - ARM7_SAVE_CFG_? - R/W mask 00FFFFFFh (reset=007FD000h=8376320)
10018124h - ARM7_SAVE_CFG_? - R/W mask 00FFFFFFh (reset=007FD000h=8376320)
10018128h - ARM7_SAVE_CFG_? - R/W mask 000FFFFFh (reset=000000E0h=224)
1001812Ch - ARM7_SAVE_CFG_? - R/W mask 000FFFFFh (reset=00051000h=331776)
Maybe video/color config, or write/erase timings, or so?
```
"This is copied from rom footer + 10h."
```
Uh, but the description of the ROM footer doesn't mention anything at "footer+10h".
Maybe this is the "ghosting" stuff, whatever that is... maybe color bleeding?

_____ Memory Maps in GBA/NDS/DSi Modes _____

Memory map in GBA Mode
```
3DS       --> GBA
08080000h --> 0E000000h, GBA Cart FLASH/SRAM/EEPROM (max 128Kbyte)
080A0000h --> 06000000h, GBA 2D-Engine VRAM (64K+32Kbyte)
080B8000h --> 03000000h, GBA Fast WRAM (32Kbyte)
080C0000h --> 02000000h, GBA Slow WRAM (256Kbyte)
20000000h --> 08000000h, GBA Cart ROM (with unstable addresses) (32Mbyte max)
```
Cart ROM is unstable: About 99% reads are reading from address+20h (within 800h-byte pages), and about 1% are randomly reading as wanted from address+0. There must be some burst-read or prefetch happening... unknown how to disable that effect?
When in GBA mode, ARM9 seems to be no longer able to access the remapped ARM9 memory blocks (ARM9 only sees 00h's)? And, ARM9/ARM11 seem to HANG (without any exception) when trying to access Main RAM at 20000000h?

Memory map in NDS/DSi Mode
```
3DS       --> NDS/DSi
08000000h --> 06800000h, NDS VRAM A (128Kbyte)
08020000h --> 06820000h, NDS VRAM B (128Kbyte)
08040000h --> 06840000h, NDS VRAM C (128Kbyte)
08060000h --> 03800000h, NDS ARM7 WRAM (64Kbyte) (ARM7 only)
08070000h --> 06898000h, NDS VRAM H (32Kbyte)
08078000h --> 068A0000h, NDS VRAM I (16Kbyte)
08080000h --> 06860000h, NDS VRAM D (128Kbyte)
080A0000h --> 06880000h, NDS VRAM E (64Kbyte)
```

```
  080B0000h --> 06890000h, NDS VRAM F (16Kbyte)
  080B4000h --> 06894000h, NDS VRAM G (16Kbyte)
  080B8000h --> 03000000h, NDS Shared RAM (32Kbyte) (initially mapped to ARM9)
  080C0000h --> 03xxxxxh, DSi New Shared WRAM A (256Kbyte) (Misc)
  10174000h --> 04804000h, NDS Wifi RAM (8Kbyte)
  1FF00000h --> 03xxxxxh, DSi New Shared WRAM B (256Kbyte) (DSP Code)
  1FF40000h --> 03xxxxxh, DSi New Shared WRAM C (256Kbyte) (DSP Data)
  20000000h --> 02000000h, NDS Main RAM (max 16MByte)     ;\only each 4th
  20000000h --> 0C000000h, DSi Main RAM (max 32Mbyte)     ;/halfword used
  ITCM/DTCM --> ITCM/DTCM, NDS ITCM/DTCM (32K+16K, same mapping as in 3DS mode)
  FF-filled --> 08000000h, GBA Cart ROM/SRAM (32MB+64K) (empty, FFh-filled)
```

Main RAM needs enable in EXMEMCNT, Main RAM uses only the lower 16bit of the 64bit FCRAM data bus (ie. only each 4th halfword is used). OAM/Palette need enable in POWCNT1. VRAM needs enable in VRAMCNT. New Shared WRAM needs MBK. Wifi RAM needs POWCNT2.

GBA Footers

GBA ROM-images are reportedly stored in .code files (in NCCH .app files, which can also contain the usual NCCH icon/banner/logo).

Nintendo is reportedly using footers in GBA ROM-images (=at the end of the .code file?). There appear to be no public dumps of that footers, but there are two homebrew descriptions, both incomplete, and conflicting with each other:

```
  3dbrew suggests ".CAA" in first 16 bytes of 360h-byte footer
  yml suggests ".CAA" in last 12 bytes of 35Ch-byte footer
```

Merging the info from that descriptions, the best guess would be that the footers might look like so:

```
 Config Data:
  000h 4    Unknown (totally crazy guess: maybe contains a 32bit value...?)
  004h 4    ROM Size (probably same as Romsize)
  008h 4    Cartridge Type (Port 10018100h, ARM7_SAVE_TYPE)
  00Ch 4    Unknown (reportedly 0000FFFFh) (wild guess: savedata fillvalue?)
  010h 4    Unknown (Port 10018120h, reportedly 1561662 or 2607238) ;\maybe
  014h 4    Unknown (Port 10018124h, reportedly 156166  or 577077 ) ; write
  018h 4    Unknown (Port 10018128h, reportedly 134     or 388    ) ; erase
  01Ch 4    Unknown (Port 1001812Ch, reportedly 187667  or 201072 ) ;/timings?
  020h 4    LCD Ghosting (01h..FFh) (uh, what is that?)
  024h 300h LCD Video LUT (guess: maybe for Port 10400484h/10400584h or so?)
  324h 0Ch  Unknown (guess: Maybe ALL bytes just Padding, probably zero)
 1st Descriptor:
  330h 4    Unknown (guess: Type 00h=ROM-Image)
  334h 4    Unknown (guess: ROM-image Offset, probably usually 0)
  338h 4    Unknown (guess: ROM-image Size, probably usually Romsize)
  33Ch 4    Unknown (guess: Padding, probably zero)
 2nd Descriptor:
  340h 4    Unknown (guess: Type 01h=Config Data)
  344h 4    Unknown (guess: Config Offset, probably usually Romsize+0)
  348h 4    Unknown (guess: Config Size, probably usually 324h or 330h or so)
  34Ch 4    Unknown (guess: Padding, probably zero)
 Footer Entrypoint (guess: probably in last 10h-byte of .code file):
  350h 4    GBA Footer ID          (".CAA")
  354h 4    Maybe Version          (must be 1)
  358h 4    Descriptor List Offset (probably usually Romsize+330h)
  35Ch 4    Descriptor List Size   (probably usually 20h) (N*10h)
```

Problem: 3dbrew says that the first descriptor must have type=01h... or more specifically, it allows type=00h as 1st descriptor... but only if next descriptor does ALSO have type=00h... which would mean that none could have type<>00h ???

GBA Savedata

Upon power-off, GBA savedata is usually temporarily stored in a small eMMC partition, and later copied to the actual savedata file upon reboot. The file format and filename for that savedata are unknown...?

ARM7 Registers Misc...

ARM7 has the GBA BIOS implemented in hardware. The BIOS is completely identical to the original GBA

BIOS (uh, actually, it is the GBA BIOS version from NDS, which has [3F0Ch]=01h).
The system is booted silently by calling SWI 01h (aka RegisterRamReset), followed by jumping to the code that does SWI 00h (aka SoftReset) to finish booting. The boot splash is still in BIOS, however, and can be seen by calling or replacing one of the previous interrupts with SWI 26h (aka HardReset).
ARM7_BOOTCODE: "This is the first code that will be run after execution begins. TwlProcess9 uses this to put ARM7 in a loop (TWL), and to set the POSTFLG and branch to more copied code (GBA)."

3DS SPI and I2C Bus

```
_____________________________________ SPI Bus _____________________________________
```

SPI Bus
3DS SPI Registers

SPI Devices
3DS SPI Devices

```
_____________________________________ I2C Bus _____________________________________
```

I2C Bus
3DS I2C Registers

I2C Device List
3DS I2C Device List

I2C Device Specs...
3DS I2C MCU Register Summary
3DS I2C MCU[00h-01h,05h-07h] - Firmware
3DS I2C MCU[02h-0Fh] - Status
3DS I2C MCU[10h-1Fh] - Interrupt Flags
3DS I2C MCU[20h-24h] - Power Control
3DS I2C MCU[28h-2Eh] - LED Control
3DS I2C MCU[40h-51h] - Accelerometer/Pedometer
3DS I2C MCU[60h-7Fh] - Misc Status
3DS I2C MCU secondary I2C Devices (on MCU bus)
DSi I2C Device 4Ah (BPTWL chip)
3DS I2C Gyroscope (old version)
3DS I2C Gyroscope (new version)
3DS I2C Infrared Receiver/Transmitter (IrDA)
3DS I2C LCD Screen Controllers
3DS I2C New3DS Near-Field Communication (NFC)
3DS I2C New3DS C-Stick and ZL/ZR-Buttons
3DS I2C New3DS 16bit IO Expander (aka QTM)
3DS I2C Other/Unused/Debug Devices

3DS SPI Registers

SPI Registers
There are four SPI buses (most of them can have up to 3 devices attached):

```
  1000D000h ARM9        SPI_CARD (savedate in game card SPI FLASH memory)
  10160000h ARM9/ARM11  SPI_BUS0 (0=Powerman, 1=Wifi-FLASH, 2=Touchscr/sound)
```

```
  10142000h ARM9/ARM11  SPI_BUS1 (0=Touchscr/sound with extra 3ds registers)
  10143000h ARM9/ARM11  SPI_BUS2 (unused)
```
The separate registers (at the above base addresses) are:
```
  10xxx000h 2   SPI_MANUAL_CNT    (R/W)  ;\Manual Access
  10xxx002h 1/2 SPI_MANUAL_DATA   (R/W)  ;/
  10xxx800h 4   SPI_FIFO_CNT      (R/W)  ;\
  10xxx804h 4   SPI_FIFO_DONE     (R/W)  ;
  10xxx808h 4   SPI_FIFO_BLKLEN   (R/W)  ; FIFO Access
  10xxx80Ch 4   SPI_FIFO_DATA     (R/W)  ; (and AUTOPOLL for
  10xxx810h 4   SPI_FIFO_STATUS   (R)    ; status reads)
  10xxx814h 4   SPI_AUTOPOLL      (R/W)  ;
  10xxx818h 4   SPI_FIFO_INT_MASK (R/W)  ;
  10xxx81Ch 4   SPI_FIFO_INT_STAT (R/ack);/
```
Interrupt IDs and DMA/FIFO startup/peripheral IDs are:
```
  SPI_CARD   ARM9 IF.bit23    Various (ARM9 NDMA and ARM9 XDMA)
  SPI_BUS0   ARM11 IRQ 56h    ARM11 CDMA 0Fh
  SPI_BUS1   ARM11 IRQ 57h    ARM11 CDMA 10h
  SPI_BUS2   ARM11 IRQ 24h    ARM11 CDMA 11h
```

Config registers
```
  CFG11_SPI_CNT - mode select for SPI_BUS0,1,2 (Manual or Fifo)
  CFG9_CARD_CTL - mode select for SPI_CARD     (Manual or Fifo or NTRCARD)
```
3DS Config - CONFIG11 Registers
3DS Config - CONFIG9 Registers

```
_____________________________ SPI Manual Access Mode ______________________________
```

10xxx000h - SPI_MANUAL_CNT
10xxx002h - SPI_MANUAL_DATA
Manual access mode works same as on DSi. Max transfer rate is 8MHz, 1bit mode only, and the manual byte-by-byte transfer adds some software overload to the transfer time.
DS Serial Peripheral Interface Bus (SPI)
One advantage is that the manual mode supports true bi-directional SPI transfers (which aren't really needed because most SPI chips use only one direction at a time, with dummy data in opposite direction).

```
_____________________________ SPI FIFO Access Mode _______________________________
```

10xxx800h - SPI_FIFO_CNT (R/W)
```
  0-2   Baudrate   (0=512KHz, 1=1MHz, 2=2MHz, 3=4MHz, 4=8MHz, 5..7=16MHz)
  3-5   Unused (0)
  6-7   SPI_BUS0-2: Device Select (0..2, or 3=Hangs/Reads as 2)
        SPI_CARD:   Unused (0)
  8-11  Unused (0)
  12    Bus Mode                (0=1bit, 1=4bit)
  13    Transfer Direction      (0=Read, 1=Write)
  14    Unused (0)
  15    Start Transfer          (0=Idle/Ready, 1=Start/Busy)
  16-31 Unused (0)
```
CARD: This register seems to have a bug where the lower 8 bits are shifted up by 16 when reading this register. Uh, really?

10xxx804h - SPI_FIFO_DONE (R/W) (or R when busy?)
```
  0     Chip Select             (0=Not Selected, 1=Selected)
  1-31  Unused (0)
```
Select occurs automatically (when starting a transfer via SPI_FIFO_CNT). However, deselect must be done manually (by writing 0 to SPI_FIFO_DONE after last data block). This allows to keep the chip selected during multiple transfer blocks (including blocks with different data direction, eg. WriteCmd + ReadData).

10xxx808h - SPI_FIFO_BLKLEN (R/W) (or R when busy?)

```
  0-20  Transfer length (1..1FFFFFh bytes, 0=?)
  21-31 Unused (0)
```

10xxx80Ch - SPI_FIFO_DATA (R/W)
```
  0-31  32bit FIFO for reading/writing data
```

10xxx810h - SPI_FIFO_STATUS (R)
```
  0     Whatever FIFO busy... or FIFO full (0=Not full, 1=Full)
  1-31  unknown/unspecified
```
"At transfer start and every 32 bytes the FIFO becomes busy."
Uh, unknown if this refers to Send FIFO or Receive FIFO. For the latter having a Empty flag would be most useful.
Might be also a generic "data request" flag, rather than a "full/busy" flag?

10xxx814h - SPI_AUTOPOLL (R/W)
```
  0-7   Command byte (eg. 05h=RDSR, aka FLASH read status)
  8-15  Unused (0)
  16-19 Timeout    (0..10 = 1ms,2ms,4ms,8ms,..,512ms,1024ms, or 11..15=Never)
  20-23 Unused (0)
  24-26 Bit number (0..6=Bit0..6, or 7=Bugged?, always "ready" after 1st poll?)
  27-29 Unused (0)
  30    Bit value  (0=WaitUntilZero, 1=WaitUntilSet)
  31    Start Poll (0=Idle/Ready, 1=Start/Busy)
```
Autopoll does repeatedly transfer command+reply bytes, until reaching the desired reply bit state (or until reaching timeout). Autopoll uses the device number and baudrate selected in SPI_FIFO_CNT.

10xxx818h - SPI_FIFO_INT_MASK (R/W) (or R when busy?)
```
  0     Transfer Finished Interrupt Disable  (0=Enable, 1=Disable)
  1     Autopoll Success Interrupt Disable   (0=Enable, 1=Disable)
  2     Autopoll Timeout Interrupt Disable   (0=Enable, 1=Disable)
  3     Unknown (R/W)                        (?)
  4-31  Unused (0)
```

10xxx81Ch - SPI_FIFO_INT_STAT (R/ack)
```
  0     Transfer Finished Flag   (0=No, 1=Yes/IRQ) (write 1 to clear)
  1     Autopoll Success Flag    (0=No, 1=Yes/IRQ) (write 1 to clear)
  2     Autopoll Timeout Flag    (0=No, 1=Yes/IRQ) (write 1 to clear)
  3     Unknown (usually 0, but might be something related to INT_MASK.bit3 ?)
  4-31  Unused (0)
```
Bit0 also fires on each autopoll try? No: it does not fire on autopoll.

3DS SPI Devices

SPI Device List
```
  Bus0, Device 0      Power Managment Device             ;\
  Bus0, Device 1      Wifi FLASH                         ; same as NDS/DSi
  Bus0, Device 2      DSi Touchscreen/Sound/Microphone   ;/
  Bus1, Device 0      3DS Touchscreen/Sound/Microphone   ;-extra 3ds registers
  Bus1, Device 1,2    Unused
  Bus2, Device 0,1,2  Unused
  Card, Device 0      Cartridge SPI (eg. FLASH savedata) ;-similar as NDS/DSi
  Card, Device 1,2    Don't exist (there are no device-select bits for card)
```

SPI Bus/Device 0:0 - Power Managment Device
The old power managment SPI registers are mostly for NDS mode:
DS Power Management Device
On 3DS, backlight and power-down are controlled via MCU.

SPI Bus/Device 0:1 - Wifi FLASH
This chip did contain the firmware in NDS consoles. On 3DS it does merely contain some wifi/user settings, mostly for NDS titles.
DS Firmware Serial Flash Memory
DS Firmware Header
DS Firmware Wifi Calibration Data
DS Firmware Wifi Internet Access Points
DS Firmware User Settings
The 3DS does probably additionally store User settings (and maybe also Wifi Acess Point info) in a eMMC file, but unknown where?

SPI Bus/Device 0:2 - DSi Touchscreen/Sound (DSi mode)
This SPI bus is used for accessing the TSC chip in DSi mode.
DSi Touchscreen/Sound Controller
Touchscreen calibration points for NDS/DSi titles are in Wifi FLASH (matched to the current NDS/DSi screen resolution, which can be 256x192pix or fullscreen; selected by holding Start+Select when starting NDS/DSi title).

SPI Bus/Device 1:0 - 3DS Touchscreen/Sound (3DS mode)
This is another SPI bus wired to the TSC chip, this bus can be accessed in 3DS mode only, giving access to the normal DSi registers, plus some additional 3DS registers in TSC page 64h,65h,67h,FBh. These new registers are required for activating sound output, and for reading the Circle Pad.
3DS TSC, Register Summary
Touchscreen calibration points for 3DS titles are stored in HWCAL0/1.dat files, that files do also include calibration data for Circle Pad, and various initial TSC register settings.

Cartridge SPI
Cartridge SPI can be wired to FLASH savedata (or theoretically anything else).
This can use 4bit SPI mode (if the chip is 4bit compatible, and if 4bit related signals are wired up accordingly, and if the specific commands support 4bit transfers).

3DS TSC, Register Summary

Page Selection
```
TSC[xxh:00h]=page    ;Page (each TSC SPI bus probably has own page+index?)
```

Page 00h-01h (DSi Registers)
```
TSC[00h:02h]=read    ;DSi Undocumented status (reserved bits)
TSC[00h:03h]=read    ;DSi Overtemperature OT Flag (reserved bits)
TSC[00h:0Bh]=87h     ;DSi DAC NDAC Value
TSC[00h:39h]=66h     ;DSi ADC DC Measurement 1 (reset=00h, ORed with 66h)
TSC[00h:3Fh]=D4h     ;DSi DAC Data-Path Setup  (reset=D4h, ORed with C0h)
TSC[00h:40h]=00h     ;DSi DAC Volume Control
TSC[00h:41h]=FDh     ;DSi DAC Left Volume Control  ;\aka 3DS     ;HWCAL[2E4h]
TSC[00h:42h]=FDh     ;DSi DAC Right Volume Control ;/ShutterVol0 ;HWCAL[2E4h]
TSC[00h:51h]=00h/80h ;DSi Microphone Mute/Unmute ;ADC Digital Mic
TSC[00h:52h]=80h/00h ;DSi Microphone Mute/Unmute ;ADC Digital Volume
TSC[01h:2Eh]=03h     ;DSi Microphone MIC BIAS
TSC[01h:2Fh]=2Bh     ;DSi Microphone MIC PGA
TSC[01h:30h]=40h     ;DSi P-Terminal ADC Channel Fine-Gain Input (reset=40h)
TSC[01h:31h]=40h     ;DSi M-Terminal ADC Input Selection         (reset=40h)
```
The 3DS does usually access only the registers mentioned above (but there are many more DSi-style registers in page 00h,01h,03h; see DSi chapter for details).

Page 04h-0Ch (DSi Coefficient RAM)

```
  TSC[04h:08h-0Dh]=... ;DSi Mic Autogain ;IIR, as DSi (7Fh,E1h,80h,1Fh,7Fh,C1h)
  TSC[05h:08h-3Fh]=... ;3DS FilterMic32  ;IIR+Biquad A,B,C,D,E;HWCAL[364h-39Bh]
  TSC[05h:48h-7Fh]=... ;3DS FilterMic47  ;IIR+Biquad A,B,C,D,E;HWCAL[39Ch-3D3h]
  TSC[08h:0Ch-3Dh]=... ;3DS FilterFreeB  ;Biquad ;\initialized;HWCAL[3DAh-40Bh]
  TSC[08h:4Ch-7Dh]=... ;3DS FilterFreeB' ;Biquad ; for        ;HWCAL[3DAh-40Bh]
  TSC[09h:02h-07h]=... ;3DS FilterFreeA  ;IIR.L  ; non-GBA    ;HWCAL[3D4h-3D9h]
  TSC[09h:08h-0Dh]=... ;3DS FilterFreeA' ;IIR.R  ;/only       ;HWCAL[3D4h-3D9h]
  TSC[0Ah:02h-07h]=... ;3DS FilterFreeA'';IIR?      HP47  ;\  ;HWCAL[3D4h-3D9h]
  TSC[0Ah:0Ch-3Dh]=... ;3DS FilterFreeB'';Biquad?   HP47  ;/  ;HWCAL[3DAh-40Bh]
  TSC[0Bh:02h-1Fh]=... ;3DS FilterHP32   ;Biquad.L A,B,C  ;\  ;HWCAL[2ECh-309h]
  TSC[0Bh:20h-3Dh]=... ;3DS FilterHP47.L ;Biquad.L D,E,F  ;/  ;HWCAL[30Ah-327h]
  TSC[0Bh:42h-5Fh]=... ;3DS FilterHP32'  ;Biquad.R A,B,C  ;\  ;HWCAL[2ECh-309h]
  TSC[0Bh:60h-7Dh]=... ;3DS FilterHP47.R ;Biquad.R D,E,F  ;/  ;HWCAL[30Ah-327h]
  TSC[0Ch:02h-1Fh]=... ;3DS FilterSP32   ;Biquad.L A,B,C  ;\  ;HWCAL[328h-345h]
  TSC[0Ch:20h-3Dh]=... ;3DS FilterSP47   ;Biquad.L D,E,F  ;/  ;HWCAL[346h-363h]
  TSC[0Ch:42h-5Fh]=... ;3DS FilterSP32'  ;Biquad.R A,B,C  ;\  ;HWCAL[328h-345h]
  TSC[0Ch:60h-7Dh]=... ;3DS FilterSP47'  ;Biquad.R D,E,F  ;/  ;HWCAL[346h-363h]
```

The above coefficient RAM pages exists on DSi, too. However, the DSi is usually initializing only those in page 04h.
Unknown how the 3DS is using the extra coefficients... does it use miniDSP instructions for that?
Unknown what the duplicated entries are for... maybe left/right channels?

Page 64h (3DS Sound/Microphone Config)

```
  TSC[64h:01h]=01h     ;3DS Software Reset (?)
  TSC[64h:22h]=18h     ;3DS ? (reset=00h, ORed with 18h, later bit2=cleared)
  TSC[64h:25h]=read    ;3DS status, wait for bit3,7
  TSC[64h:26h]=read    ;3DS status, wait for bit3,7
  TSC[64h:2Ch]         ;unused, but nonzero  ;bit0,1=headphone connect status
  TSC[64h:30h]         ;unused, but nonzero
  TSC[64h:31h]=00h/44h ;3DS ? (reset=00h) (GBA:00h, Other:44h)
  TSC[64h:43h]=11h/91h ;3DS set to 11h, later toggles bit=0 then bit7=1
  TSC[64h:44h]         ;unused, but nonzero
  TSC[64h:45h]=20h/30h ;3DS Speaker off (reset=00h, later=20h, 30h=speakerOff)
  TSC[64h:75h]         ;unused, but nonzero
  TSC[64h:76h]=14h/D4h ;3DS Lock Coefficient RAM? (reset=14h, ORed with C0h)
  TSC[64h:77h]=0Ch/00h ;3DS ? (reset=0Ch, later clear bit2,3 after coeff init)
  TSC[64h:78h]=00h     ;3DS ?
  TSC[64h:7Ah]=00h     ;3DS ?
  TSC[64h:7Bh]=ECh     ;3DS ShutterVolume1                   ;HWCAL[2E5h]
  TSC[64h:7Ch]=0Ah     ;3DS ? (reset=0Ah, later clears bit0)
```

Page 65h (3DS Sound/Microphone Gains)

```
  TSC[65h:0Ah]=0Ah     ;3DS ?
  TSC[65h:0Bh]=1Ch/3Ch ;3DS ?  ... depends on TSC[00h:02h..03h]   ;HP
  TSC[65h:0Ch]=04h     ;3DS DriverGainHP                     ;HWCAL[2E0h]*8+4
  TSC[65h:11h]=10h/D0h ;3DS ? (reset=00h, ORed with 10h, later ORed with C0h)
  TSC[65h:12h]=06h     ;3DS DriverGainSP    ;\maybe left?    ;HWCAL[2E1h]*4+2
  TSC[65h:13h]=06h     ;3DS DriverGainSP'   ;/      right?   ;HWCAL[2E1h]*4+2
  TSC[65h:16h]=00h     ;3DS AnalogVolumeHP Left  (0..7Eh?)   ;HWCAL[2E2h]
  TSC[65h:17h]=00h     ;3DS AnalogVolumeHP Right (0..7Eh?)   ;HWCAL[2E2h]
  TSC[65h:1Bh]=07h     ;3DS AnalogVolumeSP  ;\maybe left?    ;HWCAL[2E3h]
  TSC[65h:1Ch]=07h     ;3DS AnalogVolumeSP' ;/      right?   ;HWCAL[2E3h]
  TSC[65h:33h]=03h     ;3DS MicrophoneBias                   ;HWCAL[2E6h]
  TSC[65h:41h]=00h+wait;3DS PGA_GAIN (mic)      (bit0-5)     ;HWCAL[2E8h]
  TSC[65h:42h]=02h+wait;3DS QuickCharge (what?) (bit0-1)     ;HWCAL[2E7h]
  TSC[65h:47h,4Bh,4Ch,4Dh,4Eh,52h,53h]  ;unused, but nonzero
  TSC[65h:77h]=94h/95h ;3DS ? (reset=94h, ORed with 01h)
  TSC[65h:78h]         ;unused, but nonzero
  TSC[65h:7Ah]=01h     ;3DS ?
```

Some here seem to be alike TSC[1:24h]..TSC[1:2Bh]

Page 67h,FBh (3DS Touchscreen/Circle Pad)

```
  TSC[67h:17h]=43h     ;3DS AnalogSense & Precharge     ;HWCAL[40Fh]+[40Eh]*10h
```

```
TSC[67h:19h]=69h     ;3DS AnalogStabilize & XP Pullup ;HWCAL[40Dh]+[411h]*10h
TSC[67h:1Bh]=80h     ;3DS AnalogDebounce & YM_Driver  ;HWCAL[410h]+[412h]*80h
TSC[67h:24h]=98h/18h ;3DS bit7=0=touchscreen.on  ;bit2=1=has new touchdata?
TSC[67h:25h]=43h/53h ;3DS bit5-2=0100b=touchscreen.on
TSC[67h:26h]=00h/ECh ;3DS bit7=1=touchscreen.on  ;bit1=1=had old touchdata?
TSC[67h:27h]=11h     ;3DS AnalogInterval           ;HWCAL[40Ch]+10h
TSC[FBh:01h]=read    ;3DS fifo 26x16bit; 5xTSC.x, 5xTSC.y, 8xCPAD.y, 8xCPAD.x
```

3DS I2C Registers

I2C Registers

```
Address   Width Old3DS  Name
10161000h 1     Yes     I2C_BUS0_DATA     ;\
10161001h 1     Yes     I2C_BUS0_CNT      ; BUS 0 (old DSi devices)
10161002h 2     Yes     I2C_BUS0_CNTEX    ;
10161004h 2     Yes     I2C_BUS0_SCL      ;/
10144000h 1     Yes     I2C_BUS1_DATA     ;\
10144001h 1     Yes     I2C_BUS1_CNT      ; BUS 1 (extra 3DS devices)
10144002h 2     Yes     I2C_BUS1_CNTEX    ;
10144004h 2     Yes     I2C_BUS1_SCL      ;/
10148000h 1     Yes     I2C_BUS2_DATA     ;\
10148001h 1     Yes     I2C_BUS2_CNT      ; BUS 2 (extra 3DS gimmicks)
10148002h 2     Yes     I2C_BUS2_CNTEX    ;
10148004h 2     Yes     I2C_BUS2_SCL      ;/
```

10161000h - I2C_BUS0_DATA
10144000h - I2C_BUS1_DATA
10148000h - I2C_BUS2_DATA
Unknown. Supposedly data.

10161001h - I2C_BUS0_CNT
10144001h - I2C_BUS1_CNT
10148001h - I2C_BUS2_CNT

```
0     Stop  (0=No, 1=Stop/last byte)
1     Start (0=No, 1=Start/first byte)
2     Pause (0=Transfer Data, 1=Pause after Error, used with/after Stop)
3     unknown/unspecified
4     Ack Flag (0=Error, 1=Okay) (For DataRead: W, for DataWrite: R)
5     Data Direction (0=Write, 1=Read)
6     Interrupt Enable (0=Disable, 1=Enable)
7     Start/busy (0=Ready, 1=Start/busy)
```

10161002h - I2C_BUS0_CNTEX
10144002h - I2C_BUS1_CNTEX
10148002h - I2C_BUS2_CNTEX

```
0     Current SCL pin state (0=Low, 1=High/idle)                      (R)
1     Wait if SCL held low  (0=No/fixed delay, 1=Yes, wait if SCL=low) (R/W)
3-14  Unused (0)
15    Unknown (BUS0: can be set, alongsides bit1 becomes read-only?)  (?)
```

Bit1=0 appends a short fixed delay after each byte. Bit1=1 will automatically wait if the SCL clock pin is held low: If the peripheral doesn't do that then it can be even slightly faster than the fixed delay, if the peripheral does hold SCL low then the 3DS will wait as long as needed (eg. required for 3DS/New3DS MCU and New3DS C-Stick).

10161004h - I2C_BUS0_SCL
10144004h - I2C_BUS1_SCL
10148004h - I2C_BUS2_SCL

```
0-5   Duration for SCL=Low   (0..3Fh, 0=Fastest)  ;default=0
```

```
  6-7   Unused (0)
  8-12  Duration for SCL=High  (0..1Fh, 0=Fastest)  ;default=5
  13-15 Unused (0)
```
The fastest setting (0000h) is about 380kHz (about 41Kbyte/s).
The slowest setting (1F3Fh) is about 84kHz (about 9Kbyte/s)
Hmmm, or maybe the raw bitrate is faster, but with short pause between bytes?

3DS I2C Device List

I2C Devices

```
  id  bus:dev  service      Device description
  0   0:4Ah    "i2c::MCU"   BPTWL (aka MCU registers for DSi mode)  ;\same
  1   0:7Ah    "i2c::CAM"   Camera0 (internal camera, self-facing)  ; as DSi
  2   0:78h    "i2c::CAM"   Camera1 (external camera, right eye)    ;/
  3   1:4Ah    "i2c::MCU"   MCU (aka MCU registers for 3DS mode)
  4   1:78h    "i2c::CAM"   Camera2 (external camera, left eye)
  5   1:2Ch    "i2c::LCD"   Upper LCD screen (lcd0)
  6   1:2Eh    "i2c::LCD"   Lower LCD screen (lcd1)
  7   1:40h    "i2c::DEB"   Reserved for Debug?
  8   1:44h    "i2c::DEB"   Reserved for Debug?
  9   2:A6h/D6h"i2c::HID"   Debug?? addr changed from A6h to D6h in 8.0.0-18
  10  2:D0h    "i2c::HID"   Gyroscope (old version)  <-- read = FFh's ?
  11  2:D2h    "i2c::HID"   Gyroscope (new version)  <-- EXISTS in New3DS
  12  2:A4h    "i2c::HID"   Reserved for DebugPad
  13  2:9Ah    "i2c::IR"    Infrared Transmitter/Receiver (IrDA)
  14  2:A0h    "i2c::EEP"   Reserved for DebugEeprom?
  15  1:EEh    "i2c::NFC"   New3DS-only NFC (Near-field communication)
  16  0:40h    "i2c::QTM"   New3DS-only QTM (head tracking?) IO Expander!
  17  2:54h    "i2c::IR"    New3DS-only C-stick and ZL/ZR.. 44pin chip??
  -   none(?)  -            New3DS-only Hasn't co-packaged EEPROM in NFC chip?
  -   0:A0h    -            Reserved for Unknown DSi camera0 (Ext)
  -   0:E0h    -            Reserved for Unknown DSi camera1 (Self)
  -   0:40h    -            Reserved for Unknown DSi Debug stuff? and QTM
  -   0:90h    -            Reserved for Unknown DSi Debug stuff?
  -   0:00h-0Eh            Mirrors to BPTWL for whatever reason
  -   0:F0h-FEh            Mirrors to BPTWL for whatever reason
  -   0:5Ah                Internal dummy addr used by BPTWL when busy?
  -   2:00h                Unknown, something responds here with ACK and FFh's
```
Notice: These device addresses are used for writing to the respective device, for reading bit0 must be set (see I2C protocol).

3DS Secondary I2C Devices

There are also some internal/secondary I2C busses (not connected to the ARM CPUs).

```
  MCU:30h   Accelerometer                                ;\
  MCU:6Ch   Fuel Gauge MAX17040 (or newer chip in New3DS) ; connected to
  MCU:84h   video related?   ;Power Managment Device?     ; MCU chip)
  MCU:A4h   batt.flg/volume? ;Touchscreen/Sound chip?     ;/
  Wifi:A0h  I2C bus EEPROM                   ;connected to Atheros Wifi chip
  NFC:Axh   NFC chip has on-chip CAT24C64 EEPROM (?), but not wired to ARM (?)
```

3DS I2C MCU Register Summary

Bus/Device 1:4Ah - MCU

The McuIndex is usually automatically incremented after each data byte (eg. allowing to read the two Version bytes in one pass).
However, there are a few arrays (at McuIndex=29h,2Dh,4Fh,61h,7Fh), which will increment the ArrayIndex instead of the McuIndex. This happens only if the index value (in the I2C command) was pointing to the array

(eg. writing 5 bytes to index 29h will update the Power LED Array. But, writing 5 bytes to index 28h will update McuIndex 28h,29h,2Ah,2Bh,2Ch; without getting stuck at 29h, and thus updating only the 1st byte of the Power LED Array).
The extra-special case is using index 60h in the I2C command (it will increment once, then stay at index 61h).

MCU Registers

```
  00h     R    Version high (bit0-3) and hardware.type? (bit4-7)         ;\Firm
  01h     R    Version low (8bit)                                        ;/
  02h     R/W  Reset Event flags                                         ;-Stat
  03h     R/W  Top screen flicker                                        ;\
  04h     R/W  Bottom screen flicker                                     ;/
  05h-07h R/W  MCU Firmware update unlock sequence   ;ARRAY[4003h]       ;-Firm
  08h     R    Raw 3D slider position                                    ;\
  09h     R    Volume slider state (00h..3Fh) (as MCUHWC:GetSoundVolume) ;
  0Ah     R    Battery PCB Temperature Celsius (signed)                  ; Stat
  0Bh     R    Fuel Gauge Battery Percentage, msb (percent, 0..64h)      ;
  0Ch     R    Fuel Gauge Battery Percentage, lsb (percent/256, 0..FFh)  ;
  0Dh     R    Fuel Gauge Battery Voltage (in 20mV units)                ;
  0Eh     R    Sub-Device Access Flags? (bit0,1,2-3,4)                   ;
  0Fh     R    Power Status Flags                                        ;/
  10h-13h R    MCU Interrupt Flags, bit0-31 (1=IRQ, cleared after read)  ;\
  14h     R    Unused (cleared after reading, like above IRQ flags)      ;
  15h-17h R/W  Unused                                                    ; IRQs
  18h-1Bh R/W  MCU Interrupt Mask, bit0-31 (0=Enable, 1=Disable)         ;
  1Ch-1Fh R/W  Unused                                                    ;/
  20h     W    System power control (bits are 0=No change, 1=Trigger)    ;\
  21h     W    Change DSi Power Button Status register BPTWL[10h]        ;
  22h     W    Used to set LCD states (bits are 0=No change, 1=Trigger)  ; PWR
  23h     -    Unused (has a dummy write handler, but does nothing)      ;
  24h     R/W  Forced Power Off delay (0=Never, 1=Fastest, 5Dh=Insane)   ;/
  25h-26h R/W  Unused
  27h     R/W  Raw volume slider state                                   ;-
  28h     R/W  Brightness of Wifi/Power/3D LEDs                          ;\
  29h *   R/W  Power LED state + Power LED blink pattern ;ARRAY[5]       ;
  2Ah     R/W  Wifi LED state, 4 bits wide                               ; LEDs
  2Bh     R/W  Camera LED state, 4bits wide                              ;
  2Ch     R/W  3D LED state, 4 bits wide                                 ;
  2Dh *   W    Notification LED Array (4+3x20h bytes)    ;ARRAY[64h]     ;
  2Eh     R    Notification LED Status when read (1=new cycle started)   ;/
  2Fh     -    Unused (has a dummy write handler, but does nothing)
  30h     R/W  RTC Time second  (7bit)   (BCD, 00h..59h)                 ;\
  31h     R/W  RTC Time minute  (7bit)   (BCD, 00h..59h)                 ;
  32h     R/W  RTC Time hour    (6bit)   (BCD, 00h..23h)                 ;
  33h     R/W  RTC Time day of week? (3bit) (?..?, ?=Monday)             ;
                   (renesas calls this "Week 0=Sunday")                  ; RTC
                   (unknown what Nintendo is using here)                 ;
  34h     R/W  RTC Time day     (6bit)   (BCD, 01h..31h)                 ;
  35h     R/W  RTC Time month   (5bit)   (BCD, 01h..12h)                 ;
  36h     R/W  RTC Time year    (8bit)   (BCD, 00h..99h)                 ;
  37h     R/W  RTC Watch Error Correction (SUBCUD)  ;NOT leap year       ;
  38h     R/W  RTC Alarm minute (7bit)   (BCD, 00h..59h)                 ;
  39h     R/W  RTC Alarm hour   (6bit)   (BCD, 00h..23h)                 ;
  3Ah     R/W  RTC Alarm day    (6bit)   (BCD, 01h..31h) ;\maybe 0=off?  ;
  3Bh     R/W  RTC Alarm month  (5bit)   (BCD, 01h..12h) ;/              ;
  3Ch     R/W  RTC Alarm year   (8bit)   (BCD, 00h..99h)                 ;
  3Dh     R    RTC RSUBC.lsb (in 32768Hz units)         ;\range 0..7FFFh ;
  3Eh     R    RTC RSUBC.msb (latched when reading lsb) ;/(or 0..80xxh)  ;
  3Fh     W    RTC Flags (bit0=ScreenBlack?, bit1=DisableRtc32KHzOutput) ;/
  40h     R/W  Accelerometer Mode (bit0=AccelerometerOn, bit1=PedometerOn)   ;\
  41h     R/W  Accelerometer Index for Manual I2C Read via MCU[44h]          ;
  42h     R/W  Unused                                                        ;
  43h     R/W  Accelerometer Index for Manual I2C Write via MCU[44h]         ;
  44h     R/W  Accelerometer Data from/to Read/Write via MCU[41h/43h]        ;
  45h,46h R    Accelerometer Output X (lsb,msb) ;resting=+/-00xxh            ;
```

```
47h,48h R    Accelerometer Output Y (lsb,msb) ;resting=+/-00xxh            ;
49h,4Ah R    Accelerometer Output Z (lsb,msb) ;resting=-41xxh (gravity)    ;
4Bh     R/W  Pedometer Step Count, bit0-7    ;\(for the current day)       ;
4Ch     R/W  Pedometer Step Count, bit8-15   ; (uh, how/which day?)        ;
4Dh     R/W  Pedometer Step Count, bit16-23  ;/(rather total count?)       ;
4Eh     R/W  Pedometer Flags (Wr.bit0=ClearArray?, Rd.bit4=ArrayFull?)     ;
4Fh *   R    Pedometer Timestamp[6] and StepCounts[2*A8h]  ;ARRAY[6+2*A8h] ;
50h,51h R/W  Pedometer Minute,Second compare values (?)                    ;/
52h-57h R/W  Unused, except some bytes are set to fixed values once and then
58h     R/W  Volume slider calibration point for 0%   (default=36h)
59h     R/W  Volume slider calibration point for 100% (default=C9h)
5Ah     R/RW Invalid, do not use!
               on newer MCU_FIRM versions this is set to FFh once and then
               on older MCU_FIRM versions this is a read-only counter
5Bh-5Fh -    N/A (write=ignored, read=FFh)
60h *   R/W  Battery-backed RAM Index                                  ;\
61h *   R/W  Battery-backed RAM Data (200 bytes) ;ARRAY[C8h]           ; Misc
62h-7Eh -    N/A (write=ignored, read=FFh)                             ;
7Fh *   R    Various system state information ;ARRAY[09h/13h]          ;/
80h-FFh -    N/A (write=ignored, read=FFh)
```

Blurb
On old versions of MCU_FIRM none of the invalid registers are masked away by the read handler function, but are still read-only. Newer MCU_FIRM versions return the hardcoded value FF instead.
Reportedly register 3Bh (RTC Alarm month) "could be used on very old MCU_FIRM versions to upload MCU firmware if some conditions (?) are met", uh?.

3DS I2C MCU[00h-01h,05h-07h] - Firmware

MCU[00h] - Version high (bit0-3) and hardware.type? (bit4-7) (R)
MCU[01h] - Version low (8bit) (R)

```
OldMCU: Version 0.008 or lower   ;\differs on LCD bits:
NewMCU: Version 0.009 or higher  ;/backlight and "push"
```

MCU[05h-07h] - MCU Firmware update unlock sequence ;ARRAY[4003h] (R/W)
Updating MCU Firmware FLASH is done by writing 4003h bytes to MCU[05h and up]. The I2C transfer timings (and flash write timings) are automatically handled via SCL-hold. The upload takes about 2 seconds, and the console does then reboot the MCU... the backlights stay on, but I2C MCU access seems to be non-functional... for a while?
The first three bytes must be 6Ah,68h,6Ch (aka "jhl"), if the 3rd write does match (and 1st-2nd write were also correct), then the MCU switches to receiving a 4000h-byte firmware image. The image consists of two blocks:

```
1000h bytes written to flash address 0000h-0FFFh (4K)
3000h bytes written to flash address 2000h-4FFFh (12K)
```

For error checking, the image must contain three identical 9-byte "HH:MM:SS",00h timestamps:

```
0FF6h..0FFEh - 1st timestamp
2000h..2008h - 2nd timestamp
4FF6h..4FFEh - 3rd timestamp
```

The first byte of the timestamp must be other than FFh, and the third byte must be ":".
BUG: The 1st/3rd timestamp are checked before applying the firmware update, but 2nd timestamp isn't checked until AFTER applying it (then causing blinking red power led, and requiring the reflash via UART pin).
3DS I2C MCU - RL78 Flash Programming via UART
Uploading bugged code may also require reflashing via UART (the UART stuff is handled in ROM, and works safely even if the whole flash is erased).

Firmware Inner Workings
The MCU contains a RL78 processor with reflashable firmware.
3DS I2C MCU - RL78 CPU Opcode List

3DS I2C MCU - RL78 CPU Opcode Map
3DS I2C MCU - RL78 CPU Registers and Flags
3DS I2C MCU - RL78 SFR Registers (Special Function Registers) (I/O ports)
3DS I2C MCU - RL78 Misc

Firmware Nocash MCU Patches

Code Execution in Battery RAM:

```
MCU[05h-07h] = "exc"  --> Battery RAM code gets executed immediately
MCU[05h-07h] = "exq"  --> Battery RAM code gets enqueued as callback
Before execution, use MCU[61h] to upload code to Battery RAM, the RAM
can be also used to store parameters and return values, you may want to
restore the original RAM content after execution for not confusing the OS.
The ROM/RAM memory map varies for different MCU firmware versions, however,
the patched firmware provides useful ROM/RAM addresses in below tables,
code should either use relative jumps, or addresses from those tables.
```

Call Table Vectors for ROM functions:

```
00080h  Process Standard Callbacks    ;\
00082h  Add Callback AX               ; Nintendo's own stuff (not useful)
00084h  Process Enqueued Callbacks    ;/
000B4h  I2C_Receive_Block             ;\Slave I2C recv/send (can be used
000B6h  I2C_Send_Block                ;/from within callbacks only)
000B8h  FLASH Init and kill IRQs      ;\
000BAh  FLASH Erase 400h bytes        ; FLASH init/erase/write/finish
000BCh  FLASH Write max 100h bytes    ; (eg. for custom code in backup area)
000BEh  FLASH Finish 400h bytes       ;/
```

Pointers to RAM arrays:

```
F4FE8h  MCU[00h..5Ah] Registers  (5Bh bytes)  ;\
F4FEAh  MCU[61h] Battery RAM     (C8h bytes)  ; Pointers to RAM arrays
F4FECh  MCU[2Dh] Notify LED RAM  (64h bytes)  ;
F4FEEh  MCU[4Fh] Pedometer_array (150h bytes) ;/
```

Other General Patches:

```
Faster power button tap duration (0s instead laggy delay)
Faster power button hold duration (1s instead 3s)
Changed Power LED color upon holding power button (purple)
Faster shutdown after holding power button (can be 0s instead 12s)
Faster shutdown after normal power down (omitting LED fade-out delay)
```

Firmware Image/Dumping

A copy of the firmware-image can be found in the System Modules folder,

```
3ds:\title\00040130\00001f02\content\000000vv.app ;MCU
3ds:\title\00040130\20001f02\content\000000vv.app ;MCU New3DS
3ds:\title\00040130\00001f03\content\000000vv.app ;MCU Safe mode
3ds:\title\00040130\20001f03\content\000000vv.app ;MCU Safe mode New3DS
```

The .app file contains a .code file with ARM11 code (and the MCU firmware somewhere inside of that .code file, usually at file offset A078h or A07Ch; starting with the 3-byte unlock code ("jhl"), followed by the 4000h-byte firmware image).

Dumping the firmware directly from FLASH memory isn't supported. However, one could patch the firmware, and then dump the backup of the old version (that will destroy the backup of the yet older version though).

3DS I2C MCU[02h-0Fh] - Status

MCU[02h] - Reset Event flags (R/W)

2bit value, writing will mask away/"acknowledge" the event, set to 3 by mcuMainLoop on reset if reset source is "Watchdog", uh?

```
0    RTC clock value got reset to defaults          (R/ack)
1    Watchdog reset happened                        (R/ack)
2-4
5    value from BPTWL[12h].bit7 (Unknown)              (R)
```

```
  6    value from BPTWL[12h].bit0 (1=IRQ on Pwr Butt tap) (R)
  7    value from BPTWL[12h].bit1 (Unknown)               (R)
```
actually, this register returns C0h or C1h
bit0 can be cleared by writing bit0=0 (writing bit0=1 has no effect)

MCU[03h] - Top screen flicker (R/W)
MCU[04h] - Bottom screen flicker (R/W)
Whatever. Maybe from HWCAL files?

MCU[08h] - Raw 3D slider position (R)
Slider.

MCU[09h] - Volume slider state (00h..3Fh) (as MCUHWC:GetSoundVolume) (R)
Slider.

MCU[0Ah] - Battery PCB Temperature Celsius (signed) (R)
```
  0-7  Degrees Celsius (51F3h-(ADC(8)*70h))/100h  ;signed, usually 17h..19h
```

MCU[0Bh] - Fuel Gauge Battery Percentage, msb (percent, 0..64h) (R)
MCU[0Ch] - Fuel Gauge Battery Percentage, lsb (percent/256, 0..FFh) (R)
Battery pecentage, from Fuel Gauge SOC register (State Of Charge).

MCU[0Dh] - Fuel Gauge Battery Voltage (in 20mV units) (R)
Battery voltage, from upper 8bit of Fuel Gauge VCELL register.

MCU[0Eh] - Sub-Device Access Flags? (bit0,1,2-3,4) (R)
```
  0    Device 6Ch Error (Fuel Gauge)    (0=Okay, 1=Error)
  1    Device 30h Error (Accelerometer) (0=Okay, 1=Error)
  2-3  Device A4h related (Volume etc.)    ;TSC[10h]
  4    LED brightness related
  5-7  Unused
```
Whatever.

MCU[0Fh] - Power Status Flags (R)
```
  0    Unused
  1    ShellState (hinge)  (1=Shell open)              ;\
  2    Unused                                          ; mcu::RTC
  3    AdapterState (Charger connected)                ;
  4    BatteryChargeState  (1=Charging)                ;/
  5    OldMCU: Unused                                  ;\
  6    OldMCU: something (1bit) ;\back-                ; mcu::GPU
  6-5  NewMCU: something (2bit) ;/light?               ;
  7    whatever                 ;-push?                ;/
```

3DS I2C MCU[10h-1Fh] - Interrupt Flags

MCU[10h-13h] - MCU Interrupt Flags, bit0-31 (1=IRQ, cleared after read) (R)
MCU[18h-1Bh] - MCU Interrupt Mask, bit0-31 (0=Enable, 1=Disable) (R/W)
```
  0     Power button press (for 27 "ticks") aka 0.2s
  1     Power button held (for 375 "ticks" aka 3s, turns off after another 12s)
  2     HOME button press (for 5 "ticks") aka 0.04s
  3     HOME button release
  4     Wifi switch button    ;uh, WHAT "switch button" ???
  5     Shell close
  6     Shell open
  7     Fatal hardware condition? (sent when MCU gets reset by Watchdog timer)
  8     Charger removed
```

```
  9     Charger plugged in
  10    RTC alarm (when some conditions are met it's
         sent when the current day and month and year
         matches the current RTC time)
         (uh, WHICH conditions? and, does it also
         support hour:minute alarm? and supposedly also
         PLAIN hour:minute WITHOUT day/month/year?)
  11    Accelerometer I2C manual read/write done
  12    Accelerometer new XYZ sample update
  13    Battery dropped below 11%, 6% or 1% (warns at those 3 points)
  14    Battery charging stop (independent of charger state)
  15    Battery charging start
 Nonmaskable(?) interrupts, uh how/why are below "nonmaskable" ???
  16    TSC[10h].bit0=1=Whatever, or BPTWL[11h]=01h=Reset
  17    TSC[10h].bit6=1=Wheteverelse
  18-21 Unused
  22    Volume slider position change
  23    Read from BPTWL[00h] version register has occurred
  24    Video Display "push" off
  25    Video Display "push" on
  26-29 Unused
  30    set by mcu sysmodule  ;\uh, flag bits aren't set by MCU (and can't be
  31    set by mcu sysmodule  ;/set by ARM), but maybe ARM sets mask bits..?
```

3DS I2C MCU[20h-24h] - Power Control

MCU[20h] - System power control (bits here are 0=No change, 1=Trigger) (W)

```
  0     Power off
  1     Reboot (unused?)
  2     Reboot (used by mcu sysmodule and LgyBg)
  3     Used by LgyBg to power off, causes hangs in 3DS-mode
  4     NewMCU: looks like power-off type (real power off, or sleep state...?)
          "an mcu::RTC command uses this, seems to do something with the
          watchdog: Bit 4 sets a bit at a RAM address which seems to control
          the watchdog timer state, then this bit is immediately unmasked.
          This field has a bitmask of 0x0F."
  4     OldMCU: Both backlights off
  5     OldMCU: Both backlights on
  6     OldMCU: Push to LCDs off   ;\uh, what is "push" ?
  7     OldMCU: Push to LCDs on    ;/
```

Unknown how to fully power-off. Writing 01h does somewhat power-off, but the battery still runs empty after 10-20 hours. Some people execute a Wait-for-Interrupt opcode after the I2C write; not tested if that does help (and if that opcode is needed on all ARM11 CPU core(s) and ARM9 side).

MCU[21h] - Change DSi Power Button Status register BPTWL[10h] (W)

00h=No Change, other=Shuffle bits and set BPTWL[10h]

```
  0     Copied to BPTWL[10h].bit3
  1     Copied to BPTWL[10h].bit0
  2     Copied to BPTWL[10h].bit1
  3     Copied to BPTWL[10h].bit5
  4     Copied to BPTWL[10h].bit4
  5     Copied to BPTWL[10h].bit6
  6-7   Not used (except, affect the "00h=No change" thing)
```

MCU[22h] - Used to set LCD states (bits here are 0=No change, 1=Trigger) (W)

```
  0     NewMCU: Push to LCDs off   ;\uh, what is "push" ?
  1     NewMCU: Push to LCDs on    ;/
  2     NewMCU: Bottom screen backlight off;\For 2DS:
  3     NewMCU: Bottom screen backlight on ;/both screens
  4     NewMCU: Top screen backlight off   ;\For 2DS:
  5     NewMCU: Top screen backlight on    ;/No effect
```

"The rest of the bits are masked away."

MCU[24h] - Forced Power Off delay (0=Never,1=Fastest,5Dh=InsaneDefault) (R/W)

```
  0-7   Delay in 8Hz units, this value seems to be battery backed
        (the value seems to also affect the bootrom error screen)
```

3DS I2C MCU[28h-2Eh] - LED Control

MCU[28h] - Brightness of Wifi/Power/3D LEDs (R/W)

Brightness for Wifi/Power/3D LEDs (when the LEDs are on). Range 00h..FFh (default is FFh=max).

MCU[29h] - Power LED state + Power LED blink pattern ;ARRAY[5] (R/W)

1st byte (Power LED mode):

```
  00h   = fade to brightness MCU[28h] with battery check
  01h   = fade to brightness MCU[28h]
  02h   = pulsating fade on/off with battery check
  03h   = fade to brightness 00h
  04h   = instantly set brightness 00h
  05h   = instantly set brightness FFh
  06h   = Blinking RED (affects Power+Notification LEDs)
  Other = Same as 00h
```

2nd..5th bytes (optional, if any written):

```
  Power LED Blink pattern (default is 55h,55h,55h,55h)
```

MCU[2Ah] - Wifi LED state, 4 bits wide (R/W)

```
  00h = Wifi LED always off
  01h..0Fh = Wifi LED on (and blink upon traffic?)
```

MCU[2Bh] - Camera LED state, 4bits wide (R/W)

```
  00h = Camera LED always off
  01h = Camera LED slowly blinking
  02h = Camera LED always on
  03h = Camera LED set via BPTWL[31h] (DSi mode)
  04h = Camera LED flash once (then switch to 00h=off)
  05h = Camera LED off once   (then switch to 02h=on)
  06h..0Fh = Invalid (same as 00h)
```

Camera LED exists only in older 3DS, not in New3DS. The camera LED uses a digital 1bit signal (without variable brightness).

MCU[2Ch] - 3D LED state, 4 bits wide (R/W)

```
  00h = 3D LED Off (or fade-out to zero)
  01h = 3D LED On  (or fade-in/out to MCU[28h] setting)
  02h..0Fh = Same as 01h (On)
```

3D LED exists only in older 3DS, not in New3DS.

MCU[2Dh] - Notification LED Array (4+3x20h bytes) ;ARRAY[64h] (W)

```
  [2Dh.00h]      ;Delay (0..FFh = Delay 1..100h)
  [2Dh.01h]      ;Brightness/divider or so?
  [2Dh.02h]      ;Speed, some timer compare value? (FFh=none?)
  [2Dh.03h]      ;unused
  [2Dh.04h..23h] ;data Red[0..1Fh]
  [2Dh.24h..43h] ;data Green[0..1Fh]
  [2Dh.44h..63h] ;data Blue[0..1Fh]
```

"It's possible to write data here with size less than 64h, and only that portion of the pattern data will get overwritten. Writing past the size of this register seems to do nothing. Reading from this register only returns zeroes."

MCU[2Eh] - Notification LED Status when read (1=new cycle started) (R)

3DS I2C MCU[40h-51h] - Accelerometer/Pedometer

MCU Registers

```
  40h     R/W  Accelerometer Mode (bit0=AccelerometerOn, bit1=PedometerOn)   ;\
  41h     R/W  Accelerometer Index for Manual I2C Read via MCU[44h]          ;
  42h     R/W  Unused                                                        ;
  43h     R/W  Accelerometer Index for Manual I2C Write via MCU[44h]         ;
  44h     R/W  Accelerometer Data from/to Read/Write via MCU[41h/43h]        ;
  45h,46h R    Accelerometer Output X (lsb,msb) ;resting=+/-00xxh            ;
  47h,48h R    Accelerometer Output Y (lsb,msb) ;resting=+/-00xxh            ;
  49h,4Ah R    Accelerometer Output Z (lsb,msb) ;resting=-41xxh (gravity)    ;
  4Bh     R/W  Pedometer Step Count, bit0-7    ;\(for the current day)       ;
  4Ch     R/W  Pedometer Step Count, bit8-15   ; (uh, how/which day?)        ;
  4Dh     R/W  Pedometer Step Count, bit16-23  ;/(rather total count?)       ;
               An up/down movement (on whichever axis that corresponds      ;
               to gravity direction) is treated as step; however, the       ;
               first step isn't counted, so 5 continous steps would         ;
               increment the counter by 4.                                  ;
  4Eh     R/W  Pedometer Flags (Wr.bit0=ClearArray?, Rd.bit4=ArrayFull?)     ;
  4Fh *   R    Pedometer Timestamp[6] and StepCounts[2*A8h]  ;ARRAY[6+2*A8h] ;
  50h,51h R/W  Pedometer Minute,Second compare values (?)                    ;/
```

Reading the Output X/Y/Z registers would be the normal way to get the accelerometer data (if enabled in the mode register).

Pedometer (Step counter)

The Pedometer is some software feature in the MCU that converts the accelerometer data to step counters. Power-off switches the accelerometer power supply off. However, it is kept powered & can do step counting in sleep mode. Step counting is used for "Play Coins" and the pre-installed statistic tool "Omoide Kirokuchou" (probably has other name in english?).

Manual access to Accelerometer I2C registers

The manual read/write feature allows to view/modify I2C registers directly; for debugging/hacking purposes or so. The manual read/write seem to be done in a callback function (=not immediately after writing MCU[41h] or MCU[44h]), and, before touching further accelerometer registers, one must wait until the callback has executed (which is indicated by MCU.interrupt.bit11).

Accelerometer Chip

On Old3DS it is marked "2048, 33DH, X1MAQ", which is said to be ST LIS331DLH.
On New3DSXL it is marked "KXTKK, 40860, .3413", which might be a Kionix chip, resembling KXTIK, but with LIS331DLH-style register map & device address.
Both Old3DS and New3DSXL have the chip on the mainboard, somewhere underneath of the bottom-screen. That (and the exact location) may be important when interpreting the accelerometer data (ie. the chip isn't parallel with the top-screen unless the case is fully unfolded).
The Accelerometer chip is wired to the MCU, so it's I2C registers can be accessed only indirectly, through the MCU.

```
_____________________ Internal Registes (for manual access) _____________________
```

Internal LIS331DLH Accelerometer I2C Register map (for manual access)

```
  Addr    Type Default  Name
  00h-0Eh  -  -        -              Reserved (do not modify)
  0Fh      R  00110010 WHO_AM_I       Device identification register (32h)
  10h-1Fh  -  -        -              Reserved (do not modify)
```

```
20h      RW 00000111 CTRL_REG1       Power Control
21h      RW 00000000 CTRL_REG2       Filter Control
22h      RW 00000000 CTRL_REG3       Interrupt Control
23h      RW 00000000 CTRL_REG4       Misc Control
24h      RW 00000000 CTRL_REG5       Sleep to Wake Control
25h      R  -        HP_FILTER_RESET High-Pass Filter Clear (Read=clear)
26h      RW 00000000 REFERENCE       High-pass Filter Reference value
27h      R  00000000 STATUS_REG      Output Status
28h      R  output   OUT_X_L         Out.X.L ;\usually near +/-00xxh
29h      R  output   OUT_X_H         OUT.X.H ;/
2Ah      R  output   OUT_Y_L         OUT.Y.L ;\usually near +/-00xxh
2Bh      R  output   OUT_Y_H         OUT.Y.H ;/
2Ch      R  output   OUT_Z_L         OUT.Z.L ;\usually near -41xxh (gravity)
2Dh      R  output   OUT_Z_H         OUT.Z.H ;/
2Eh-2Fh  -  -        -               Reserved (do not modify)
30h      RW 00000000 INT1_CFG        Interrupt 1 Config      ;\
31h      R  00000000 INT1_SOURCE     Interrupt 1 Status      ; INT 1
32h      RW 00000000 INT1_THS        Interrupt 1 Threshold   ;
33h      RW 00000000 INT1_DURATION   Interrupt 1 Duration    ;/
34h      RW 00000000 INT2_CFG        Interrupt 2 Config      ;\
35h      R  00000000 INT2_SOURCE     Interrupt 2 Status      ; INT 2
36h      RW 00000000 INT2_THS        Interrupt 2 Threshold   ;
37h      RW 00000000 INT2_DURATION   Interrupt 2 Duration    ;/
38h-3Fh  -  -        -               Reserved (do not modify)
40h-7Fh  -  -        -               Undocumented (zero)
80h-FFh              Same as 00h-7Fh, with auto-incrementing index
```

ACCL[0Fh] - WHO_AM_I - Device identification register (R)

```
0-7  Fixed (32h for LIS331DLH)
```

ACCL[20h] - CTRL_REG1 - Power Control (R/W)

```
0   Xen  X axis enable        (0=Disable, 1=Enable)              (default=1)
1   Yen  Y axis enable        (0=Disable, 1=Enable)              (default=1)
2   Zen  Z axis enable        (0=Disable, 1=Enable)              (default=1)
3-4 DR   Data rate selection  (0=50Hz, Others: see Table 20)     (default=0)
5-7 PM   Power mode selection (0=Power-down, Other: see Table 19) (default=0)
```

PM bits allow to select between power-down and two operating active modes. The device is in power-down mode when PD bits are set to "000" (default value after boot). Table 19 shows all the possible power mode configurations and respective output data rates. Output data in the low-power modes are computed with low-pass filter cut-off frequency defined by DR1, DR0 bits.

DR bits, in the normal-mode operation, select the data rate at which acceleration samples are produced. In low-power mode they define the output data resolution. Table 20 shows all the possible configuration for DR1 and DR0 bits.

Table 19. Power mode and low-power output data rate configurations:

```
PM       Power mode selection     Output data rate ODR'LP
00h      Power-down               --
01h      Normal mode              <ODR>
02h      Low-power                0.5Hz
03h      Low-power                1Hz
04h      Low-power                2Hz
05h      Low-power                5Hz
06h      Low-power                10Hz
```

Table 20. Normal-mode output data rate configurations and low-pass cut-off frequencies:

```
DR       Output Data Rate ODR     Low-pass filter cut-off frequency
00h      50Hz                     37Hz
01h      100Hz                    74Hz
02h      400Hz                    292Hz
03h      1000Hz                   780Hz
```

ACCL[21h] - CTRL_REG2 - Filter Control (R/W)

```
0-1 HPCF  High pass filter cut-off frequency (HPc) (0..3 = 8,16,32,64)
```

```
  2   HPen1 High pass filter for interrupt 1 src (0=Bypassed, 1=Filter enabled)
  3   HPen2 High pass filter for interrupt 2 src (0=Bypassed, 1=Filter enabled)
  4   FDS   Filtered data selection (0=Internal filter bypassed, 1=Data from
                                     internal filter sent to output register)
  5-6 HPM   High pass filter mode   (0=Normal mode, Other=see Table 23)
  7   BOOT  Reboot memory content   (0=Normal mode, 1=Reboot memory content)
```

BOOT bit is used to refresh the content of internal registers stored in the flash memory block. At the device power up the content of the flash memory block is transferred to the internal registers related to trimming functions to permit a good behavior of the device itself.
If for any reason the content of trimming registers was changed it is sufficient to use this bit to restore correct values. When BOOT=1 the content of internal flash is copied inside corresponding internal registers and it is used to calibrate the device. These values are factory trimmed and they are different for every accelerometer. They permit a good behavior of the device and normally they have not to be changed. At the end of the boot process the BOOT bit is set again to 0.

Table 23. High-pass filter mode configuration

HPM	High-pass filter mode
00h	Normal mode (reset reading HP_RESET_FILTER)
01h	Reference signal for filtering
02h	Normal mode (reset reading HP_RESET_FILTER)

HPCF[1:0]. These bits are used to configure high-pass filter cut-off frequency ft which is given by:

```
  ... XXX ...
```

The equation can be simplified to the following approximated equation:

```
  ... XXX ...
```

Table 24. High-pass filter cut-off frequency configuration

HPcoeff2,1	ft [Hz] Data rate=50Hz	ft [Hz] Data rate=100Hz	ft [Hz] Data rate=400Hz	ft [Hz] Data rate=1000Hz
00	1	2	8	20
01	0.5	1	4	10
10	0.25	0.5	2	5
11	0.125	0.25	1	2.5

ACCL[22h] - CTRL_REG3 - Interrupt Control (R/W)

```
  0-1  I1_CFG Data signal on INT 1 pad control bits (see table below)
  2    LIR1   Latch IRQ on INT1_SRC register, with INT1_SRC cleared by reading
                INT1_SRC itself (0=IRQ not latched, 1=IRQ latched)
  3-4  I2_CFG Data signal on INT 2 pad control bits (see table below)
  5    LIR2   Latch IRQ on INT2_SRC register, with INT2_SRC cleared by reading
                INT2_SRC itself (0=IRQ not latched, 1=IRQ latched)
  6    PP_OD  Interrupt Push-pull/Open drain    (0=Push-pull, 1=Open drain)
  7    IHL    Interrupt active high/low         (0=Active high, 1=Active low)
```

Table 27. Data signal on INT 1 and INT 2 pad

I#_CFG	INT1/INT2 Pad
00h	Interrupt 1/2 source
01h	Interrupt 1 source OR interrupt 2 source
02h	Data ready
03h	Boot running

ACCL[23h] - CTRL_REG4 - Misc Control (R/W)

```
  0    SIM    SPI serial interface mode (0=4-wire, 1=3-wire)
  1    ST     Self-test enable          (0=No, 1=Self-test)
  2    -      Reserved (0)
  3    STsign Self-test sign            (0=Self-test Plus; 1=Self-test Minus)
  4-5  FS     Full-scale selection      (0..3 = +/-2g, +/-4g, Reserved, +/-8g)
  6    BLE    Big/little endian data selection (0=Litte-endian, 1=Big-endian)
  7    BDU    Block data update (0=Continuos update, 1=Output registers
                                  not updated between MSB and LSB reading)
```

BDU bit is used to inhibit output registers update between the reading of upper and lower register parts. In default mode (BDU=0) the lower and upper register parts are updated continuously. If it is not sure to read faster than output data rate, it is recommended to set BDU=1. In this way, after the reading of the lower (upper) register part, the content of that output registers is not updated until the upper (lower) part is read too.

This feature avoids reading LSB and MSB related to different samples.

ACCL[24h] - CTRL_REG5 - Sleep to Wake Control (R/W)

```
  0-1  Turn-on mode selection for sleep to wake function (0 or 3)
  2-7  Reserved (0)
```

TurnOn bits are used for turning on the sleep to wake function.

```
  Table 32. Sleep to wake configuration
  TurnOn  Sleep to wake status
  00h     Sleep to wake function is disabled
  03h     Turned on: The device is in low power mode (ODR=Defined in CTRL_REG1)
```

Setting TurnOn=3 the "sleep to wake" function is enabled. When an interrupt event occurs the device is turned to normal mode increasing the ODR to the value defined in CTRL_REG1. Although the device is in normal mode, CTRL_REG1 content is not automatically changed to "normal mode" configuration.

ACCL[25h] - HP_FILTER_RESET - High-Pass Filter Clear content (R=clear)

Dummy register. Reading at this address zeroes instantaneously the content of the internal high pass-filter. If the high pass filter is enabled all three axes are instantaneously set to 0g.
This allows to overcome the settling time of the high pass filter.

ACCL[26h] - REFERENCE - High-pass Filter Reference value (R/W)

```
  0-7  Reference value for high-pass filter. Default=00h
```

This register sets the acceleration value taken as a reference for the high-pass filter output.
When filter is turned on (at least one of FDS, HPen2, or HPen1 bit is equal to 1) and HPM bits are set to "01", filter out is generated taking this value as a reference.

ACCL[27h] - STATUS_REG - Output Status (R)

```
  0  XDA    New X axis new data available          (0=No, 1=Available)
  1  YDA    New Y axis new data available          (0=No, 1=Available)
  2  ZDA    New Z axis new data available          (0=No, 1=Available)
  3  ZYXDA  New set of X,Y,Z axis available        (0=No, 1=Available)
  4  XOR    Overrun has overwritten X axis data    (0=No, 1=Overrun)
  5  YOR    Overrun has overwritten Y axis data    (0=No, 1=Overrun)
  6  ZOR    Overrun has overwritten Z axis data    (0=No, 1=Overrun)
  7  ZYXOR  Overrun has overwritten X,Y,Z axis data (0=No, 1=Overrun)
```

Overrun means that previous data got overwritten before it was read.
Unknown if the Status bits get cleared after reading the Status register, or after reading the Output registers, or elsewhere?

ACCL[28h,29h] - OUT_X_L, OUT_X_H - X-axis acceleration data (R)
ACCL[2Ah,2Bh] - OUT_Y_L, OUT_Y_H - Y-axis acceleration data (R)
ACCL[2Ch,2Dh] - OUT_Z_L, OUT_Z_H - Z-axis acceleration data (R)

```
  0-15  The value is expressed as two's complement.
```

ACCL[30h] - INT1_CFG - Interrupt 1 source Configuration (R/W)
ACCL[34h] - INT2_CFG - Interrupt 2 source Configuration (R/W)

```
  0  XLIE Enable interrupt generation on X low event  (0=Disable, 1=Enable)
  1  XHIE Enable interrupt generation on X high event (0=Disable, 1=Enable)
  2  YLIE Enable interrupt generation on Y low event  (0=Disable, 1=Enable)
  3  YHIE Enable interrupt generation on Y high event (0=Disable, 1=Enable)
  4  ZLIE Enable interrupt generation on Z low event  (0=Disable, 1=Enable)
  5  ZHIE Enable interrupt generation on Z high event (0=Disable, 1=Enable)
  6  6D   6 direction detection function enable.      (See table below)
  7  AOI  AND/OR combination of interrupt events.     (See table below)
```

The six X/Y/Z high/low events trigger when measured accel. value is higher/lower than preset threshold.
Interrupt mode configuration

```
  AOI 6D  Interrupt mode
  0   0   OR combination of interrupt events
  0   1   6 direction movement recognition
```

```
  1   0   AND combination of interrupt events
  1   1   6 direction position recognition
```

ACCL[31h] - INT1_SRC - Interrupt 1 source Status (R)
ACCL[35h] - INT2_SRC - Interrupt 2 source Status (R)
```
  0  XL  X low event has occurred           (0=No, 1=Yes)
  1  XH  X high event has occurred          (0=No, 1=Yes)
  2  YL  Y low event has occurred           (0=No, 1=Yes)
  3  YH  Y high event has occurred          (0=No, 1=Yes)
  4  ZL  Z low event has occurred           (0=No, 1=Yes)
  5  ZH  Z high event has occurred          (0=No, 1=Yes)
  6  IA  Interrupt active (0=No, 1=One or more interrupts have been generated)
  7  -   Reserved (0)
```
Reading at this address clears INTn_SRC.bit6 (and the interrupt signal on INTn pin) and allows the refreshment of data in the INTn_SRC register if the latched option was chosen.

ACCL[32h] - INT1_THS - Interrupt 1 threshold (R/W)
ACCL[36h] - INT2_THS - Interrupt 1? threshold (R/W)
```
  0-6  THS  Interrupt threshold. Default value: 000 0000
  7    -    Reserved (0)
```

ACCL[33h] - INT1_DURATION - Minimum duration of Interrupt 2? event (R/W)
ACCL[37h] - INT2_DURATION - Minimum duration of Interrupt 2 event (R/W)
```
  0-6  D    Duration value. Default value: 000 0000
  7    -    Reserved (0)
```
These bits set the minimum duration of the Interrupt 2(?) event to be recognized. Duration time steps and maximum values depend on the ODR chosen.

3DS I2C MCU[60h-7Fh] - Misc Status

MCU[60h] - Battery-backed RAM Index (R/W)
MCU[61h] - Battery-backed RAM Data (200 bytes) ;ARRAY[C8h] (R/W)
This is a battery-backed 200-byte general purpose RAM area. The MCU itself doesn't care about the RAM content. Instead, the ARM CPU can use it to store powerdown/error flags (and then check those flags on next boot).
```
 The 1st byte at array[00h] is used to store flags for managing FIRM/NS state:
  bit0 = "WirelessDisabled"
  bit1 = "SoftwareClosed"
  bit2 = "PowerOffInitiated"
  bit4 = "LegacyJumpProhibited"
```
When exceeding the C8h-byte array size: Writing to index=C8h..FFh is ignored (and the index is not incremented). Reading from index=C8h..FFh returns data=00h (and the index is incremented, and can wrap from index=FFh to index=00h).

MCU[7Fh] - Various system state information ;ARRAY[09h/13h] (R)
```
  [7Fh:00h] Value 00h..06h (console model? critical_hw_state?) (usually 00h ?)
  [7Fh:01h] Powerman Version (00h=normal) (from POW[00h])
  [7Fh:02h]     battery scheme? (0..7, or FFh) (maybe from middle-batt-pin?)
  [7Fh:03h] Fuel Gauge Version.msb      (00h) ;FUEL[02h]
  [7Fh:04h] Fuel Gauge Version.lsb      (12h) ;FUEL[03h]
  [7Fh:05h] Fuel Gauge Config.msb RCOMP (5Eh) ;FUEL[0Ch]
  [7Fh:06h] Battery PCB Temperature raw ADC(8) value; see MCU[0Ah] for celsius
  [7Fh:07h] Battery PCB Temperature flags (bit0=0=Bad, bit1=0=VeryBad?)
  [7Fh:08h] Fixed 01h on New3DS-XL (older 3DS can be 00h or 01h = what?)
 On MCU_FIRM major version 0:
  [7Fh:09h and up] unknown
 On MCU_FIRM major version 1:
  [7Fh:09h and up] unused (AAh)
```

```
On MCU_FIRM major version 2 and up (or so):
 [7Fh:09h] Sys Model (0=3ds, 1=3dsXL, 2=New3ds, 3=2ds, 4=New3dsXL, 5=New2dsXL)
 [7Fh:0Ah] Power LED color         (0=Blue/off, 1=Red)
 [7Fh:0Bh] Power LED intensity     (00h..FFh)
 [7Fh:0Ch] 3D LED intensity        (00h..FFh)
 [7Fh:0Dh] RGB LED red intensity   (00h..FFh)
 [7Fh:0Eh] RGB LED green intensity (00h..FFh)
 [7Fh:0Fh] RGB LED blue intensity  (00h..FFh)
 [7Fh:10h] Camera LED state        (0=Off, 1=On)
 [7Fh:11h] Wifi LED intensity      (00h..FFh)
 [7Fh:12h] Raw button states
   bit0: Power button        (0=Pressed)
   bit1: Home button         (0=Pressed)
   bit2: Wifi slider(what??) (0=Held/Pressed??)
   bit5: Charger LED         (0=LED On, 1=LED Off)  ;\or vice-versa?
   bit6: Charger connected AND busy (0=Busy, 1=No)  ;/or one just "connected?"
   this byte is reset to 0 before an svcBreak takes effect, uh?
 [7Fh:13h and up] unused (FFh)
```

3DS I2C MCU secondary I2C Devices (on MCU bus)

Bus/Device MCU:30h - Accelerometer
See Accel chapter.
3DS I2C MCU[40h-51h] - Accelerometer/Pedometer

Bus/Device MCU:6Ch - Fuel Gauge

```
 index_____________dir_;MAX17040_____________;MAX17048___;Richtek RT9428___
 FUEL[02h]         R   ;VCELL, voltage      ;VCELL      ;VBAT
 FUEL[04h]         R   ;SOC, StateOfCharge  ;SOC        ;SOC
 FUEL[06h]         W   ;MODE                ;MODE       ;CONTROL
 FUEL[08h]         R   ;VERSION             ;VERSION    ;DEVICE ID
 FUEL[0Ah]         -   ;-                   ;HIBRT      ;Status, dSOC
 FUEL[0Ch]         R/W ;RCOMP               ;CONFIG     ;CONFIG
 FUEL[0Eh]         R/W ;???                 ;???        ;OCV (but read-only!)
 FUEL[14h]         -   ;-                   ;VALRT      ;-
 FUEL[16h]         -   ;-                   ;CRATE      ;-
 FUEL[18h]         -   ;-                   ;VRESET/ID  ;-
 FUEL[1Ah]         -   ;-                   ;STATUS     ;-
 FUEL[3Eh]         W   ;???                 ;???        ;???
 FUEL[40h+0..3Eh] W   ;???                 ;TABLE      ;???
 FUEL[80h+0..1Eh] W   ;???                 ;???        ;???
 FUEL[FEh]         -   ;COMMAND             ;CMD        ;MFA
```

The fuel gauge registers are 16bit, big-endian (eg. "write(device,index,msb,lsb)").
One should always read/write 2 bytes. Trying to read a single byte from odd index does instead return the msb from even index. Trying to read more than 2 bytes does weirdly return each 2-byte pair TWICE, and does then advance to the next 2-byte pair.
Old3DS uses a small 8pin fuel gauge with marking 17040, which is apparently Maxim MAX17040.
New3DS uses a very tiny 8pin bga fuel gauge with marking 7048, which might be Maxim MAX17048 (although the MCU firmware uses some undocumented registers that aren't mentioned in MAX17048 datasheet). The MCU firmware seems to detect different fuel gauge versions/revisions, unknown which different chip(s) are used (eg. Richtek RT9428 would be also compatible).

Bus/Device MCU:84h - Power Managment Device

```
 POW[00h]  Powerman version? (can be read via MCU[7Fh:01h])
 POW[01h]  set to 00h,0Fh,1Fh
 POW[02h]  flags (bit0..bit4 or so)
 POW[03h]  flags (bit0, bit1)
 POW[04h]  backlight enable flags (bit0, bit1)
 POW[05h]  set to 00h,08h,27h   ;00h=Old3DS, 08h=New3DS, 27h=PowerOff??
```

```
  POW[06h]  from MCU[03h] ;top_screen_flicker
  POW[07h]  from MCU[04h] ;bottom_screen_flicker
  POW[08h..FFh] unused (mirrors of POW[00h..07h])
```
One should always read/write only 1 byte, trying to read more bytes returns FFh's for the extra bytes.

Bus/Device MCU:A4h - Touchscreen/Sound Controller
```
  TSC[10h]  flags?
  TSC[12h]  batt.low.flag (bit0)
  TSC[13h]  volume (38h..7Fh)
  TSC[20h]  set to AAh
```
One can read 1 or more bytes.

3DS I2C MCU - RL78 Flash Programming via UART

Setting Flash Programming Mode
```
  VDD      __/""""""""""""""""""""""""""""""
  /RESET   ______/""""""""""""""""""""""""""
  FLMD0    xx__/""""""""""""""""""""""""""" (high=flash mode)
  TOOL0    xx__/"""""""\_/"""""\_/"\_/""""" (merged uart rx/tx on one pin)
```
The 3DS seems to have wired some/all of the above pins to the test points covered by green solder stop layer.

Old3DS Mainboard (near headphone socket, all 6 pins under solderstop)
```
     ??   FLMD0  ??
    TP79  TP75  TP74
     _ |   _     _
    (_)'  (_)-  (_)-
     _     _     _
   -(_)   (_)-  (_)-
    TP78  TP76  TP77
    GND   TOOL1 TOOL0
```

New3DS-XL Mainboard (near headphone socket, right 4 pins under solderstop)
```
                      _   _
 _     _     _       TOOL0(_) (_)FLMD0
(_)   (_)   (_)            _ x _
   ____     _    _       TOOL1(_) (_)/RESET
  (____|   (_)  (_)
   VDD18 PVDD18 GND
```

Wiring Programmer (PC Parallel Port, GPIO, UART, whatever) to 3DS MCU
```
  /RESET ----------------|>|------------ PC.Data1.Reset
  TOOL0  ------------o---|>|------------ PC.Data0.TX
                     |  BAT85   |  .---- PC.Busy.RX
  FLMD0  -----.      |          | / B
              |      '--[33K]---|<  C
  PVDD18 -----'                | \ E
                          BC547 |  v--.
  GND    ----------------------------o- PC.Ground
```
The 3DS MCU programming signals are 1.8V, two BAT85 Schottky diodes can be used as one-directional level changer (with replies ignored/skipped via hardcoded delays), an additional 10K-65K resistor and BC547 transistor can be used for bi-directional transfers.

Command Summary
```
  00h Reset (aka enter flash mode)
  13h Verify/compare
  14h --Undocumented-- verify/compare first 4 bytes of flash, if [000C3h].bit1
  19h --Undocumented-- resembles "internal verify" from write command
  20h Chip Erase
  22h Block Erase
```

```
  32h Block Blank check
  40h Write
  9Ah Set Baud Rate
  A0h Set Security
  B0h Get Checksum
  C0h Get Silicon Signature
  C5h Get Version
```
The relevant commands are Reset, Block Erase, and Write.

Response Status/Error Codes
```
  04h Command Number Error
  05h Parameter Error
  06h Normal Acknowledge (ACK) (okay)
  07h Checksum Error
  0Fh Verify Error
  10h Protect Error
  15h Negative Acknowledge (NACK)
  1Ah MRG10 Error (mrg, uh?)
  1Bh MRG11 Error (mrg, uh?)
  1Ch Write Error
  FFh N/A (for 2nd status byte, when 1st status byte was bad)
```

Command Frames (Command to chip)
```
  00h 1     Command Start (fixed=01h)
  01h 1     Length (LEN) (1..255, or 00h=256)
  02h 1     Command Number
  03h LEN-1 Command Parameters
  xxh 1     Checksum (01h minus all of the above bytes)
  xxh 1     Command End (fixed=03h)
```

Data Frames (Data to chip, and Data/Status from chip)
```
  00h 1     Data Start (fixed=02h)
  01h 1     Length (LEN) (1..255, or 00h=256)
  02h LEN   Data
  xxh 1     Checksum (02h minus all of the above bytes)
  xxh 1     Data End (03h=Last Data Frame, or 17h=More data frames follow)
```

Reset Command (aka Start/Sync)
Drag FLDM0 high (1.8V in 3DS) to enable programming mode, and drag /RESET low for a moment. Then exchange the following bytes/packets.
Low Pulse (from chip, should arrive within 10ms after releasing /RESET):
```
  00h 1     Low pulse (value 00h)  ;-transferred at 9600 bps
```
Low Pulses (to chip):
```
  00h 1     Low pulse (value 00h)  ;\chip autodetects baudrate based on
  01h 1     Low pulse (value 00h)  ;/transfer time for these two bytes
```
Command (to chip):
```
  00h 3     Start/Len/Command (01h,01h,00h)
  03h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code
  03h 2     Checksum/End      (CHK,03h)
```

Set Baudrate Command
Allows to set the baudrate, which isn't really needed because the baudrate is autodetected when sending the Reset pulses.
Recommended might be 57.6Kbps, higher rates aren't really useful because of the slow write duration.
Tested/working rates are 9.6Kbps through 115.2Kbps, including odd nonstandard rates.
Bytes should be send as 8N2 (8 databits, no parity, 2 stopbits). However, the chip seems to send only 1 stopbit. Nethertheless, it seems to insist on receiving 2 stopbits (alternately, one could send the 2nd stopbit before

startbit).
Command (to chip):
```
  00h 3     Start/Len/Command (01h,06h,9Ah)
  03h 1     Sync/Connection Mode (00h=Microcontroller, 1=Programmer)
  04h 2     Baud rate, in whatever units
  06h 1     Noise filter (00h=Off, 01h=On)
  07h 1     Speed/voltage (00h=Fast 2.7V and up, 01h=Slow 1.8V and up)
  08h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code
  03h 2     Checksum/End      (CHK,03h)
```

Chip Erase Command
Erases the while flash memory & security settings (unless the security settings have disabled Chip Erase command itself).
Command (to chip):
```
  00h 3     Start/Len/Command (01h,01h,20h)
  03h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (chip erase result)
  03h 2     Checksum/End      (CHK,03h)
```

Block Erase Command
Blocks seem to be 1Kbyte (400h bytes). Erasing 1000h bytes at once does also work.
Command (to chip):
```
  00h 3     Start/Len/Command (01h,07h,22h)
  03h 3     Start address (MSB,MID,LSB) ;lower 10bit=000h  ;\400h byte boundary
  06h 3     End address   (MSB,MID,LSB) ;lower 10bit=3FFh  ;/
  09h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (block erase result)
  03h 2     Checksum/End      (CHK,03h)
```

Block Blank Check Command
Command (to chip):
```
  00h 3     Start/Len/Command (01h,08h,32h)
  03h 3     Start address (MSB,MID,LSB) ;lower 10bit=000h  ;\400h byte boundary
  06h 3     End address   (MSB,MID,LSB) ;lower 10bit=3FFh  ;/
  09h 1     Type (00h=Before single block, 01h=Before chip erase)
  0Ah 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (block blank check result)
  03h 2     Checksum/End      (CHK,03h)
```

Write Command
Requires prior Erase command, max length for erase/write commands seems to be 1000h bytes. To quit flash mode & resume normal operation, it seems to be required to unplug battery supply for moment after having written all data.
Command (to chip):
```
  00h 3     Start/Len/Command (01h,07h,40h)
  03h 3     Start address (MSB,MID,LSB) ;lower 8bit=00h  ;\100h byte boundary
  06h 3     End address   (MSB,MID,LSB) ;lower 8bit=FFh  ;/
  09h 2     Checksum/End      (CHK,03h)
```
Response (from chip, after command):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
```

```
  03h 2     Checksum/End      (CHK,03h)
```
Data (to chip):
```
  00h 2     Start/Len (02h,LEN) ;LEN=1..255 bytes, or 0=256 bytes
  02h LEN   Data (usually 256 bytes)
  xxh 2     Checksum/End      (CHK,03h/17h)  ;03h=Last, 17h=Nonlast
```
Response (from chip, circa 60ms after each data block):
```
  00h 2     Start/Len (02h,02h)
  02h 1     Status/Error code (data reception result)
  02h 1     Status/Error code (write result)
  04h 2     Checksum/End      (CHK,03h)
```
Response (from chip, extra response, about 1.5 seconds after LAST data block):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (internal verify result)
  03h 2     Checksum/End      (CHK,03h)
```
Note: The ugly 1.5 seconds verify delay can avoided by issuing a chip reset instead of waiting for the final response.

Verify/Compare Command

Command (to chip):
```
  00h 3     Start/Len/Command (01h,07h,13h)
  03h 3     Start address (MSB,MID,LSB) ;lower 8bit=00h  ;\100h byte boundary
  06h 3     End address   (MSB,MID,LSB) ;lower 8bit=FFh  ;/
  09h 2     Checksum/End      (CHK,03h)
```
Response (from chip, after command):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
  03h 2     Checksum/End      (CHK,03h)
```
Data (to chip):
```
  00h 2     Start/Len (02h,LEN)  ;LEN must be N*4 bytes (usually (1)00h bytes)
  02h LEN   Data
  xxh 2     Checksum/End      (CHK,03h/17h)  ;03h=Last, 17h=Nonlast
```
Response (from chip, after each data block):
```
  00h 2     Start/Len (02h,02h)
  02h 1     Status/Error code (data reception result)
  02h 1     Status/Error code (verify result) (always ACK/okay for Nonlast?)
  04h 2     Checksum/End      (CHK,03h)
```
With data LEN=4, this seems to allow dump the chip content via brute-force (but requires to send/receive trillions of bits to dump 4 bytes, and gets slower towards end of 100h-byte snippets).

Get Checksum Command

Command (to chip):
```
  00h 3     Start/Len/Command (01h,07h,B0h)
  03h 3     Start address (MSB,MID,LSB) ;lower 8bit=00h  ;\100h byte boundary
  06h 3     End address   (MSB,MID,LSB) ;lower 8bit=FFh  ;/blah: "from top 1KB"
  09h 2     Checksum/End      (CHK,03h)
```
Response (from chip, after command):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
  03h 2     Checksum/End      (CHK,03h)
```
Data (from chip, after above response):
```
  00h 2     Start/Len (02h,02h)
  02h 2     Data checksum, 0000h minus all data bytes (MSB,LSB)
  04h 2     Checksum/End      (CHK,03h)
```

Set Security Command

Allows write protect flash areas. Don't touch (or use with care), some settings cannot be undone (not even via Chip Erase command).

Command (to chip):
```
  00h 3     Start/Len/Command (01h,03h,A0h)
  03h 2     Fixed (00h,00h)
```

```
  08h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
  03h 2     Checksum/End      (CHK,03h)
```
Data (to chip):
```
  00h 2     Start/Len (02h,06h)   ;uh, len=6 (but below should have len=8)?
  02h 1     FLG   Security Flags (can disable write/erase etc.)
  03h 1     BOT   Boot cluster last block number ("fixed to 03h")
  04h 2     FSWS  Flash shield window start ("Higher bits, Lower bits")
  06h 2     FSWE  Flash shield window end   ("Higher bits, Lower bits")
  08h 2     FFH   Fixed (FFh,FFh)
  0Ah 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (security write result)
  03h 2     Checksum/End      (CHK,03h)
```
Response (from chip):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (internal verify result)
  03h 2     Checksum/End      (CHK,03h)
```

Get Silicon Signature Command

Command (to chip):
```
  00h 3     Start/Len/Command (01h,01h,C0h)
  03h 2     Checksum/End      (CHK,03h)
```
Response (from chip, after command):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
  03h 2     Checksum/End      (CHK,03h)
```
Data (from chip, after above response):
```
  00h 2     Start/Len (02h,1Bh)
  02h 1     VEN  Vendor                          (10h=NEC/Renseas) ;bit7=parity
  03h 1     MET  Macro extension code            (EFh)             ;bit7=parity
  04h 1     MSC  Macro function code             (04h)             ;bit7=parity
  05h 3     DET  Device extension code 1,2,3     (DCh,FDh,xxh)     ;bit7=parity
  08h 3     UAE  User flash size-1 (LSB,MID,MSB) (007FFFh, little-endian!)
  0Bh 10    DEV  Device name                     ("D79F0104  ", ASCII)
  15h 1     SCF  Security flag information                          ;\same as
  16h 1     BOT  Boot block number ("03h=fixed")                    ; from
  17h 2     FSWS Flash shield window start("Higher side, Lower side"); security
  19h 2     FSWE Flash shield window end  ("Higher side, Lower side"); command?
  1Bh 2     RES  Reserved (FFFFh)                                   ;/
  1Dh 2     Checksum/End      (CHK,03h)
```

Get Version Command

Command (to chip):
```
  00h 3     Start/Len/Command (01h,01h,C5h)
  03h 2     Checksum/End      (CHK,03h)
```
Response (from chip, after command):
```
  00h 2     Start/Len (02h,01h)
  02h 1     Status/Error code (command reception result)
  03h 2     Checksum/End      (CHK,03h)
```
Data (from chip, after above response):
```
  00h 2     Start/Len (02h,06h)
  02h 3     Device Version    (MSB,MID,LSB) (000000h)
  05h 3     Firmware Version  (MSB,MID,LSB) (010000h)
  08h 2     Checksum/End      (CHK,03h)
```

Notes

The above protocol is documented in the "78K0R/Kx3-L Flash Memory Programming" application note.

Unknown if verify/checksum could be tweaked to dump the current flash content. There is also an undocumented anti-dumping feature related to bytes at 000C4h (that's probably related to TOOL1 debug pin, not to TOOL0 serial flash pin).

3DS I2C MCU - RL78 CPU Opcode List

Move Byte

```
09/29/49                 MOV A, [nnnn+B]/[nnnn+C]/[nnnn+BC]
88/89/8A/8B/8C           MOV A, [nn+SP]/[DE]/[nn+DE]/[HL]/[nn+HL]
8D/8E/8F                 MOV A, [saddr]/[sfr]/[nnnn]
61C9/61E9                MOV A, [HL+B]/[HL+C]
60/--/62/63/64/65/66/67  MOV A, X/-/C/B/E/D/L/H
70/--/72/73/74/75/76/77  MOV X/-/C/B/E/D/L/H, A
50/51/52/53/54/55/56/57  MOV X/A/C/B/E/D/L/H, imm8
D8/--/F8/E8              MOV X/-/C/B, [saddr]
D9/--/F9/E9              MOV X/-/C/B, [nnnn]
E0/E1/E2/E3/E4/E5        MOV X/A/C/B/[saddr]/[nnnn], 1  ;native: ONEB
F0/F1/F2/F3/F4/F5        MOV X/A/C/B/[saddr]/[nnnn], 0  ;native: CLRB
41/61B8                  MOV ES, imm8/[saddr]
18/28/48                 MOV [nnnn+B]/[nnnn+C]/[nnnn+BC], A
98/99/9A/9B/9C           MOV [nn+SP]/[DE]/[nn+DE]/[HL]/[nn+HL], A
9D/9E/9F                 MOV [saddr]/[sfr]/[nnnn], A
61D9/61F9                MOV [HL+B]/[HL+C], A
19/38/39                 MOV [nnnn+B]/[nnnn+C]/[nnnn+BC], imm8
C8/CA/CC                 MOV [nn+SP]/[nn+DE]/[nn+HL], imm8
CD/CE/CF                 MOV [saddr]/[sfr]/[nnnn], imm8
08/--/618A/618B/618C/618D/618E/618F   XCH A, X/-/C/B/E/D/L/H
61AC/61AD/61B9/61A9                   XCH A, [HL]/[HL+byte]/[HL+B]/[HL+C]
61AE/61AF                             XCH A, [DE]/[DE+byte]
61AB/61A8/61AA                        XCH A, [sfr]/[saddr]/[nnnn]
```

Move Word

```
13/15/17/30              MOVW AX, BC/DE/HL/imm16
59/69/79                 MOVW AX, [nnnn+B]/[nnnn+C]/[nnnn+BC]
A8/A9/AA/AB/AC           MOVW AX, [nn+SP]/[DE]/[nn+DE]/[HL]/[nn+HL]
AD/AE/AF                 MOVW AX, [saddrp]/[sfrp]/[nnnn]
E6/E7                    MOVW AX/BC,1    ;native: ONEW AX/BC
F6/F7                    MOVW AX/BC,0    ;native: CLRW AX/BC
12/32/DA/DB              MOVW BC, AX/#imm16/[saddrp]/[nnnn]
14/34/EA/EB              MOVW DE, AX/#imm16/[saddrp]/[nnnn]
16/36/FA/FB              MOVW HL, AX/#imm16/[saddrp]/[nnnn]
58/68/78                 MOVW [nnnn+B]/[nnnn+C]/[nnnn+BC], AX
B8/B9/BA/BB/BC           MOVW [nn+SP]/[DE]/[nn+DE]/[HL]/[nn+HL] ,AX
BD/BE/BF                 MOVW [saddrp]/[sfrp]/[nnnn], AX
C9/CB                    MOVW [saddrp]/[sfrp], imm16
C0/C2/C4/C6/61CD         POP  AX/BC/DE/HL/PSW
C1/C3/C5/C7/61DD         PUSH AX/BC/DE/HL/PSW
33/35/37                 XCHW AX, BC/DE/HL
```

ALU Byte

```
6100/6101/6102/6103/6104/6105/6106/6107  ADD  X/A/C/B/E/D/L/H, A
6110/6111/6112/6113/6114/6115/6116/6117  ADDC X/A/C/B/E/D/L/H, A
6120/6121/6122/6123/6124/6125/6126/6127  SUB  X/A/C/B/E/D/L/H, A
6130/6131/6132/6133/6134/6135/6136/6137  SUBC X/A/C/B/E/D/L/H, A
6140/6141/6142/6143/6144/6145/6146/6147  CMP  X/A/C/B/E/D/L/H, A
6150/6151/6152/6153/6154/6155/6156/6157  AND  X/A/C/B/E/D/L/H, A
6160/6161/6162/6163/6164/6165/6166/6167  OR   X/A/C/B/E/D/L/H, A
6170/6171/6172/6173/6174/6175/6176/6177  XOR  X/A/C/B/E/D/L/H, A
6108/----/610A/610B/610C/610D/610E/610F  ADD  A, X/-/C/B/E/D/L/H
6118/----/611A/611B/611C/611D/611E/611F  ADDC A, X/-/C/B/E/D/L/H
6128/----/612A/612B/612C/612D/612E/612F  SUB  A, X/-/C/B/E/D/L/H
```

```
6138/----/613A/613B/613C/613D/613E/613F  SUBC A, X/-/C/B/E/D/L/H
6148/----/614A/614B/614C/614D/614E/614F  CMP  A, X/-/C/B/E/D/L/H
6158/----/615A/615B/615C/615D/615E/615F  AND  A, X/-/C/B/E/D/L/H
6168/----/616A/616B/616C/616D/616E/616F  OR   A, X/-/C/B/E/D/L/H
6178/----/617A/617B/617C/617D/617E/617F  XOR  A, X/-/C/B/E/D/L/H
0B/0C/0D/0E/0F                ADD  A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
1B/1C/1D/1E/1F                ADDC A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
2B/2C/2D/2E/2F                SUB  A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
3B/3C/3D/3E/3F                SUBC A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
4B/4C/4D/4E/4F                CMP  A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
5B/5C/5D/5E/5F                AND  A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
6B/6C/6D/6E/6F                OR   A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
7B/7C/7D/7E/7F                XOR  A,[saddr]/#byte/[HL]/[HL+byte]/[nnnn]
0A                            ADD  [saddr],#byte
1A                            ADDC [saddr],#byte
2A                            SUB  [saddr],#byte
3A                            SUBC [saddr],#byte
4A/40                         CMP  [saddr]/[nnnn],#byte
5A                            AND  [saddr],#byte
6A                            OR   [saddr],#byte
7A                            XOR  [saddr],#byte
D0/D1/D2/D3/D4/D5              CMP  X/A/C/B/[saddr]/[nnnn],0  ;native: CMP0
80/81/82/83/84/85/86/87/A0/A4/6159 INC X/A/C/B/E/D/L/H/[nnnn]/[saddr]/[HL+nn]
90/91/92/93/94/95/96/97/B0/B4/6169 DEC X/A/C/B/E/D/L/H/[nnnn]/[saddr]/[HL+nn]
```

ALU Word

```
01/03/05/07             ADDW AX, AX/BC/DE/HL
--/23/25/27             SUBW AX, --/BC/DE/HL
--/43/45/47             CMPW AX, --/BC/DE/HL
04/02/06/6109/10        ADDW AX, nnnn/[nnnn]/[saddrp]/[HL+nn] / ADDW SP,00nn
24/22/26/6129/20        SUBW AX, nnnn/[nnnn]/[saddrp]/[HL+nn] / SUBW SP,00nn
44/42/46/6149/--        CMPW AX, nnnn/[nnnn]/[saddrp]/[HL+nn] / CMPW --
A1/A3/A5/A7/A2/A6/6179  INCW AX/BC/DE/HL/[nnnn]/[saddrp]/[HL+nn]
B1/B3/B5/B7/B2/B6/6189  DECW AX/BC/DE/HL/[nnnn]/[saddrp]/[HL+nn]
```

Rotate/Shift

```
61DB                  ROR A, 1
61EB                  ROL A, 1
61FB                  RCR A, 1         ;native: RORC A,1
61DC                  RCL A, 1         ;native: ROLC A,1
3109/3108/3107+n*10   SHL A/B/C, 1..7
310A+n*10             SHR A, 1..7
310B+n*10             SAR A, 1..7
61EE/61FE             RCLW AX/BC, 1    ;native: ROLWC AX/BC,1
310D/310C+n*10        SHLW AX/BC, 1..15
310E+n*10             SHRW AX, 1..15
310F+n*10             SARW AX, 1..15
```

Jump/Call

```
61CB                  JMP AX                  ;native: BR AX             ;CS:
ECnnnn0n              JMP absolute far addr   ;native: BR !!addr20
EDnnnn                JMP absolute addr       ;native: BR !addr16
EEnnnn                JMP relative addr       ;native: BR $!addr20
EFnn                  JMP relative short addr ;native: BR $addr20
61CA/61DA/61EA/61FA   CALL AX/BC/DE/HL        ;native: CALL AX/BC/DE/HL  ;CS:
FCnnnn0n              CALL absolute far addr  ;native: CALL !!addr20
FDnnnn                CALL absolute addr      ;native: CALL !addr16
FEnnnn                CALL relative addr      ;native: CALL $!addr20
61nn                  CALL [000xx]            ;native: CALLT [xx]
```

Conditional Relative Jumps

```
DCnn                  JC  aka JB  addr  ;carry/below      ;native: BC
DDnn                  JZ  aka JE  addr  ;zero/equal       ;native: BZ
```

```
61C3nn                JH  aka JA  addr  ;higher/above     ;native: BH
DEnn                  JNC aka JAE addr  ;not carry/below  ;native: BNC
DFnn                  JNZ aka JNE addr  ;not zero/equal   ;native: BNZ
61D3nn                JNH aka JBE addr  ;not higher/above ;native: BNH
```

Test (and Clear) and Conditional Relative Jump

```
3100/3180/3181/3101+n*10h  BTCLR|JNZCLR [saddr]/[sfr]/[HL]/A.n, addr
3102/3182/3183/3103+n*10h  BT|JNZ       [saddr]/[sfr]/[HL]/A.n, addr
3104/3184/3185/3105+n*10h  BF|JZ        [saddr]/[sfr]/[HL]/A.n, addr
```

Bit Manipulation

```
7101/7109/7181/7189+n*10            MOV1 [saddr]/[sfr]/[HL]/A.n, CY
7104/710C/7184/718C+n*10            MOV1 CY, [saddr]/[sfr]/[HL]/A.n
7105/710D/7185/718D+n*10            AND1 CY, [saddr]/[sfr]/[HL]/A.n
7106/710E/7186/718E+n*10            OR1  CY, [saddr]/[sfr]/[HL]/A.n
7107/710F/7187/718F+n*10            XOR1 CY, [saddr]/[sfr]/[HL]/A.n
7180/7100/7102/710A/7182/718A+n*10  SET1 CY/[nnnn]/[saddr]/[sfr]/[HL]/A.n
7188/7108/7103/710B/7183/718B+n*10  CLR1 CY/[nnnn]/[saddr]/[sfr]/[HL]/A.n
71C0                                NOT1 CY
```

Misc

```
00                              NOP
61CC/61ED/61FD                  BRK/HALT/STOP
D7/61EC/61FC                    RET/RETB/RETI
FF                              TRAP     ;pseudo invalid.reset
717AFA/717BFA                   EI/DI    ;pseudo set/clr.sfr.bit
61C8/61D8/61E8/61F8/61E3/61F3   SKC/SKNC/SKZ/SKNZ/SKH/SKNH  ;conditional skip
61CE                            MOVS [HL+byte],X
61DE                            CMPS X,[HL+byte]
61CF/61DF/61EF/61FF             SEL RB0/1/2/3
D6                              MULU X   ;unsigned multiply AX=A*X
11                              prefix for [ES:addr]   ;native: ES:addr[reg]
```

3DS I2C MCU - RL78 CPU Opcode Map

Instruction Map (1st Map) (without prefix)

	00	10	20	30
00	NOP	ADDW SP,#byte	SUBW SP,#byte	MOVW AX,#word
01	ADDW AX,AX	PREFIX (ES:)	-	4th MAP !!!
02	ADDW AX,!addr16	MOVW BC,AX	SUBW AX,!addr16	MOVW BC,#word
03	ADDW AX,BC	MOVW AX,BC	SUBW AX,BC	XCHW AX,BC
04	ADDW AX,#word	MOVW DE,AX	SUBW AX,#word	MOVW DE,#word
05	ADDW AX,DE	MOVW AX,DE	SUBW AX,DE	XCHW AX,DE
06	ADDW AX,saddrp	MOVW HL,AX	SUBW AX,saddrp	MOVW HL,#word
07	ADDW AX,HL	MOVW AX,HL	SUBW AX,HL	XCHW AX,HL
08	XCH A,X	MOV word[B],A	MOV word[C],A	MOV word[C],#byte
09	MOV A,word[B]	MOV word[B],#byte	MOV A,word[C]	MOV word[BC],#byte
0A	ADD saddr,#byte	ADDC saddr,#byte	SUB saddr,#byte	SUBC saddr,#byte
0B	ADD A,saddr	ADDC A,saddr	SUB A,saddr	SUBC A,saddr
0C	ADD A,#byte	ADDC A,#byte	SUB A,#byte	SUBC A,#byte
0D	ADD A,[HL]	ADDC A,[HL]	SUB A,[HL]	SUBC A,[HL]
0E	ADD A,[HL+byte]	ADDC A,[HL+byte]	SUB A,[HL+byte]	SUBC A,[HL+byte]
0F	ADD A,!addr16	ADDC A,!addr16	SUB A,!addr16	SUBC A,!addr16

	40	50	60	70
40	CMP !addr16,#byte	MOV X,#byte	MOV A,X	MOV X,A
41	MOV ES,#byte	MOV A,#byte	2nd MAP !!!	3rd MAP !!!
42	CMPW AX,!addr16	MOV C,#byte	MOV A,C	MOV C,A
43	CMPW AX,BC	MOV B,#byte	MOV A,B	MOV B,A
44	CMPW AX,#word	MOV E,#byte	MOV A,E	MOV E,A
45	CMPW AX,DE	MOV D,#byte	MOV A,D	MOV D,A

```
 46  CMPW AX,saddrp     MOV  L,#byte     MOV  A,L         MOV  L,A
 47  CMPW AX,HL         MOV  H,#byte     MOV  A,H         MOV  H,A
 48  MOV  word[BC],A    MOVW word[B],AX  MOVW word[C],AX  MOVW word[BC],AX
 49  MOV  A,word[BC]    MOVW AX,word[B]  MOVW AX,word[C]  MOVW AX,word[BC]
 4A  CMP  saddr,#byte   AND  saddr,#byte OR   saddr,#byte XOR  saddr,#byte
 4B  CMP  A,saddr       AND  A,saddr     OR   A,saddr     XOR  A,saddr
 4C  CMP  A,#byte       AND  A,#byte     OR   A,#byte     XOR  A,#byte
 4D  CMP  A,[HL]        AND  A,[HL]      OR   A,[HL]      XOR  A,[HL]
 4E  CMP  A,[HL+byte]   AND  A,[HL+byte] OR   A,[HL+byte] XOR  A,[HL+byte]
 4F  CMP  A,!addr16     AND  A,!addr16   OR   A,!addr16   XOR  A,!addr16

     80                 90               A0               B0
 80  INC  X             DEC  X           INC  !addr16     DEC  !addr16
 81  INC  A             DEC  A           INCW AX          DECW AX
 82  INC  C             DEC  C           INCW !addr16     DECW !addr16
 83  INC  B             DEC  B           INCW BC          DECW BC
 84  INC  E             DEC  E           INC  saddr       DEC  saddr
 85  INC  D             DEC  D           INCW DE          DECW DE
 86  INC  L             DEC  L           INCW saddrp      DECW saddrp
 87  INC  H             DEC  H           INCW HL          DECW HL
 88  MOV  A,[SP+byte]   MOV  [SP+byte],A MOVW AX,[SP+byte] MOVW [SP+byte],AX
 89  MOV  A,[DE]        MOV  [DE],A      MOVW AX,[DE]     MOVW [DE],AX
 8A  MOV  A,[DE+byte]   MOV  [DE+byte],A MOVW AX,[DE+byte] MOVW [DE+byte],AX
 8B  MOV  A,[HL]        MOV  [HL],A      MOVW AX,[HL]     MOVW [HL],AX
 8C  MOV  A,[HL+byte]   MOV  [HL+byte],A MOVW AX,[HL+byte] MOVW [HL+byte],AX
 8D  MOV  A,saddr       MOV  saddr,A     MOVW AX,saddrp   MOVW saddrp,AX
 8E  MOV  A,sfr         MOV  sfr,A       MOVW AX,sfrp     MOVW sfrp,AX
 8F  MOV  A,!addr16     MOV  !addr16,A   MOVW AX,!addr16  MOVW !addr16,AX

     C0                 D0               E0               F0
 C0  POP  AX            CMP0 X           ONEB X           CLRB X
 C1  PUSH AX            CMP0 A           ONEB A           CLRB A
 C2  POP  BC            CMP0 C           ONEB C           CLRB C
 C3  PUSH BC            CMP0 B           ONEB B           CLRB B
 C4  POP  DE            CMP0 saddr       ONEB saddr       CLRB saddr
 C5  PUSH DE            CMP0 !addr16     ONEB !addr16     CLRB !addr16
 C6  POP  HL            MULU X ;(AX=A*X) ONEW AX          CLRW AX
 C7  PUSH HL            RET              ONEW BC          CLRW BC
 C8  MOV  [SP+byte],#byte MOV  X,saddr   MOV  B,saddr     MOV  C,saddr
 C9  MOVW saddrp,#word  MOV  X,!addr16   MOV  B,!addr16   MOV  C,!addr16
 CA  MOV  [DE+byte],#byte MOVW BC,saddrp MOVW DE,saddrp   MOVW HL,saddrp
 CB  MOVW sfrp,#word    MOVW BC,!addr16  MOVW DE,!addr16  MOVW HL,!addr16
 CC  MOV  [HL+byte],#byte BC   $addr20    BR   !!addr20    CALL !!addr20
 CD  MOV  saddr,#byte   BZ   $addr20     BR   !addr16     CALL !addr16
 CE  MOV  sfr,#byte     BNC  $addr20     BR   $!addr20    CALL $!addr20
 CF  MOV  !addr16,#byte BNZ  $addr20     BR   $addr20     - (TRAP)
```

Opcode FFh = TRAP (illegal opcode, used by 3DS MCU, triggers Reset vector).
Uh, other/prefixed illegal opcodes do not act as TRAP?

Instruction Map (2nd MAP) (with prefix byte 61h)

```
     00    10    20    30    40    50    60    70      Notes
 00  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands X,A
 01  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,A
 02  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands C,A
 03  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands B,A
 04  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands E,A
 05  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands D,A
 06  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands L,A
 07  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands H,A
 08  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,X
 09  ADDW  -     SUBW  -     CMPW  INC   DEC   INCW    <see below>
 0A  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,C
 0B  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,B
 0C  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,E
```

```
  0D  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,D
  0E  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,L
  0F  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,H
  80  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,[HL+B]
  81  -     -     -     -     -     -     -     -       -
  82  ADD   ADDC  SUB   SUBC  CMP   AND   OR    XOR     with operands A,[HL+C]
  83  -     -     -     -     BH    BNH   SKH   SKNH    <see below>
  84  [80h] [82h] [84h] [86h] [88h] [8Ah] [8Ch] [8Eh]   CALLT [0008xh]
  85  [90h] [92h] [94h] [96h] [98h] [9Ah] [9Ch] [9Eh]   CALLT [0009xh]
  86  [A0h] [A2h] [A4h] [A6h] [A8h] [AAh] [ACh] [AEh]   CALLT [000Axh]
  87  [B0h] [B2h] [B4h] [B6h] [B8h] [BAh] [BCh] [BEh]   CALLT [000Bxh]
  88  -     -     XCH   MOV   SKC   SKNC  SKZ   SKNZ    <see below>
  89  DECW  -     XCH   XCH   MOV   MOV   MOV   MOV     <see below>
  8A  XCH   -     XCH   -     CALL  CALL  CALL  CALL    <see below>
  8B  XCH   -     XCH   -     BR    ROR   ROL   RORC    <see below>
  8C  XCH   -     XCH   -     BRK   ROLC  RETB  RETI    <see below>
  8D  XCH   -     XCH   -     POP   PUSH  HALT  STOP    <see below>
  8E  XCH   -     XCH   -     MOVS  CMPS  ROLWC ROLWC   <see below>
  8F  XCH   -     XCH   -     SEL   SEL   SEL   SEL     <see below>
```

Operands for above opcodes:

```
  Operands for ADDW/SUBW/CMPW                AX,[HL+byte]
  Operands for INCW/DECW/INC/DEC             [HL+byte]
  Operands for BR                            AX
  Operands for ROL/ROR/ROLC/RORC             A,1
  Operands for PUSH/POP                      PSW
  Operands for MOVS                          [HL+byte],X
  Operands for CMPS                          X,[HL+byte]
  Operands for SKC/SKNC/SKZ/SKNZ/SKH/SKNH    None
  Operands for BRK/RETB/RETI/HALT/STOP       None
  Operands for BH/BNH                        $addr20
  Operands for ROLWC (opcode EE/FE)          AX,1 / BC,1
  Operands for CALL  (opcode CA/DA/EA/FA)    AX/BC/DE/HL
  Operands for SEL   (opcode CF/DF/EF/FF)    RB0/RB1/RB2/RB3
  Operands for MOV   (opcode C9/D9)          A,[HL+B] / [HL+B],A
  Operands for MOV   (opcode E9/F9/B8)       A,[HL+C] / [HL+C],A / ES,saddr
  Operands for XCH   (opcode 8A..8F)         A,C / A,B / A,E / A,D / A,L / A,H
  Operands for XCH   (opcode A8,A9,AA)       A,saddr / A,[HL+C] / A,!addr16
  Operands for XCH   (opcode AB,AC,AD)       A,sfr /  A,[HL] / A,[HL+byte]
  Operands for XCH   (opcode AE,AF,B9)       A,[DE] / A,[DE+byte] / A,[HL+B]
```

Note Not mounted on the RL78-S1 core.

Instruction Map (3rd MAP) (with prefix byte 71h)

```
  00+#*10  SET1 !addr16.#          80+#*10  --> see below (80,C0)
  01+#*10  MOV1 saddr.#,CY         81+#*10  MOV1 [HL].#,CY
  02+#*10  SET1 saddr.#            82+#*10  SET1 [HL].#
  03+#*10  CLR1 saddr.#            83+#*10  CLR1 [HL].#
  04+#*10  MOV1 CY,saddr.#         84+#*10  MOV1 CY,[HL].#
  05+#*10  AND1 CY,saddr.#         85+#*10  AND1 CY,[HL].#
  06+#*10  OR1  CY,saddr.#         86+#*10  OR1  CY,[HL].#
  07+#*10  XOR1 CY,saddr.#         87+#*10  XOR1 CY,[HL].#
  08+#*10  CLR1 !addr16.#          88+#*10  --> see below (88)
  09+#*10  MOV1 sfr.#,CY           89+#*10  MOV1 A.#,CY
  0A+#*10  SET1 sfr.#              8A+#*10  SET1 A.#
  0B+#*10  CLR1 sfr.#              8B+#*10  CLR1 A.#
  0C+#*10  MOV1 CY,sfr.#           8C+#*10  MOV1 CY,A.#
  0D+#*10  AND1 CY,sfr.#           8D+#*10  AND1 CY,A.#
  0E+#*10  OR1  CY,sfr.#           8E+#*10  OR1  CY,A.#
  0F+#*10  XOR1 CY,sfr.#           8F+#*10  XOR1 CY,A.#
  80       SET1 CY                 90,A0,B0             reserved
  88       CLR1 CY                 D0,E0,F0             reserved
  C0       NOT1 CY                 98,A8,B8,C8,D8,E8,F8 reserved
```

Instruction Map (4th MAP) (with prefix byte 31h)

```
  00+#*10  BTCLR saddr.#,$addr20 ;\
```

```
01+#*10  BTCLR A.#,    $addr20 ;
80+#*10  BTCLR sfr.#,  $addr20 ;
81+#*10  BTCLR [HL].#, $addr20 ;
02+#*10  BT    saddr.#,$addr20 ; #=0..7
03+#*10  BT    A.#,    $addr20 ;
82+#*10  BT    sfr.#,  $addr20 ;
83+#*10  BT    [HL].#, $addr20 ;
04+#*10  BF    saddr.#,$addr20 ;
05+#*10  BF    A.#,    $addr20 ;
84+#*10  BF    sfr.#,  $addr20 ;
85+#*10  BF    [HL].#, $addr20 ;/
06+#*10  -                     ;-#=N/A (0..15=reserved)
07+#*10  SHL   C,#             ;\
08+#*10  SHL   B,#             ;
09+#*10  SHL   A,#             ; #=1..7 (0,8..15=reserved)
0A+#*10  SHR   A,#             ;
0B+#*10  SAR   A,#             ;/
0C+#*10  SHLW  BC,#            ;\
0D+#*10  SHLW  AX,#            ; #=1..15 (0=reserved)
0E+#*10  SHRW  AX,#            ;
0F+#*10  SARW  AX,#            ;/
```

Pseudo Opcodes

```
EI         71 7A FA         ;SET1 SFR(FAh).7  (PSW.bit7 enable interrupts)
DI         71 7B FA         ;CLR1 SFR(FAh).7  (PSW.bit7 disable interrupts)
TRAP       FF               ;invalid opcode FF (triggers Reset vector)
```

Extra MUL opcodes (unknown if these are supported on 3DS):

```
MULHU      CE FB 01         ;MOV SFR(FBh),#01h    Note
MULH       CE FB 02         ;MOV SFR(FBh),#02h    Note
DIVHU      CE FB 03         ;MOV SFR(FBh),#03h    Note
DIVWU      CE FB 0B         ;MOV SFR(FBh),#0Bh    Note
MACHU      CE FB 05         ;MOV SFR(FBh),#05h    Note
MACH       CE FB 06         ;MOV SFR(FBh),#06h    Note
```

Others:

```
"saddr.#" and "sfr.#" with #=8..15 could be implemented as "addr+1.#-8".
"callt [nn]" can be seen as 2-byte alias for "call nnnn" (via the ROM table)
```

3DS I2C MCU - RL78 CPU Registers and Flags

Registers

```
PC   20bit   Program Counter
PSW  8bit    Program Status Word (flags)
SP   16bit   Stack Pointer (bit0=always 0, bit1-15=used)
AX   16bit   aka A:X   ;\these can be used as 16bit (eg. AX) or two 8bit
BC   16bit   aka B:C   ; registers (eg. A and X).
DE   16bit   aka D:E   ; all of these registers exist in 4 banks (the
HL   16bit   aka H:L   ;/active bank is selected via "SEL RBn" opcode)
ES   4bit    Data Bank ;-used only for PREFIX'ed opcode, otherwise bank=0Fh
CS   4bit    Code Bank ;-MSBs of destination for JMP/CALL AX/BC/DE/HL
```

Note: All registers (except PC) are mapped to the SFR register area, and can be also used by accessing that memory. In case of the banked AX/BC/DE/HL registers, observe that direct memory access won't recurse the current "bank" selection (that weird feature/problem is same as for 8051 CPUs).

RL78 Memory Map

```
00000h 80h    Code Vector Table                                  ;\
00080h 40h    Code Callt Table                                   ;
000C0h 4      Code Option byte area (4 bytes)                    ; Code
000C4h 0Ah    Code On-chip debug security ID setting area        ; FLASH
000CEh F32h   Code                                               ; memory
01000h 1000h  Code (can be hardware-swapped with 00000h..00FFFh?)  ;
```

```
02000h 3000h  Code                                                ;
05000h 3000h  Code (3DS uses this area as firmware backup copy)   ;/
08000h 8000h  Unused, FFh-filled (does NOT seem to be flash)
10000h DD800h Unused, 00h-filled
ED800h 2800h  ROM area (or A4h,FBh,5Ah,FAh-filled when disabled)  ;\
EFFD0h 2      ROM data, FLASH (size-1)/100h   ;007Fh=32Kbyte      ; ROM
EFFF8h 4      ROM code, jump vector for flash functions (c=func)  ;/
F0000h 7E0h   Special-function registers (2nd SFRs)
F07E0h 620h   Hidden RAM (for use by ROM) (or 00h-filled when disabled) ;-RAM
F0E00h 200h   Mirror of RAM at FF900h     (or 00h-filled when disabled)
F1000h 7000h  Mirror of FLASH ROM code area at 01000h (for [Fxxxxh] data)
F8000h 78F0h  Mirror of unused FFh-filled area at 08000h
FF8F0h 10h    Hidden RAM (for use by ROM) (or FFh-filled when disabled) ;-?
FF900h 520h   RAM
FFE20h C0h    RAM (C0h bytes, short-addressable)         ;short addr 20h..DFh
FFEE0h 20h    Registers AX,BC,DE,HL (in four banks)      ;short addr E0h..FFh
FFF00h 20h    SFRs (Port 0-15, etc.)  ;SFR addr 00h..1Fh ;short addr 00h..1Fh
FFF20h D8h    SFRs                    ;SFR addr 20h..F7h
FFFF8h 8      Registers SP,PSW,etc.   ;SFR addr F8h..FFh
```

RL78 Program Status Word (PSW aka Flags)

```
    S1    S2/S3
0   CY    CY    Carry flag (0=No, 1=Carry/borrow)
1   ISP0  ISP0  In-service Priority bit0
2   ISP1  ISP1  In-service Priority bit1
3   0     RBS0  Register Bank Select bit0   (RL78-S2/S3 only) (used on 3DS)
4   AC    AC    Auxilliary Carry flag (aka carry on 4bit nibble)
5   0     RBS1  Register Bank Select bit1   (RL78-S2/S3 only) (used on 3DS)
6   Z     Z     Zero flag             (0=No, 1=Zero/Equal)
7   IE    IE    Interrupt Enable flag (0=Disable, 1=Enable)
```

RL78 Opcode flags

```
Opcode                                ZHC
ADD/ADDC/SUB/SUBC/CMP/CMPS            XXX
ADDW/SUBW/CMPW ;with dst<>SP          XXX
ADDW/SUBW      ;with dst=SP           ---
CMP0                                  X00
AND/OR/XOR                            X--
INC/DEC                               XX-
INCW/DECW                             ---
MOVS                                  X-X
SHR/SHRW/SHL/SHLW/SAR/SARW            --X
ROR/RORC/ROL/ROLC/ROLWC               --X
AND1/OR1/XOR1/NOT1                    --X
MOV1                                  ---
MOV/XCH/ONEB/CLRB                     ---
MOVW/XCHW/ONEW/CLRW                   ---
MULU                                  ---
SET1/CLR1                             ---
CALL/CALLT/BRK/RET/PUSH/POP           ---
BR/Bcond/SKcond/BTCLR                 ---
SEL/NOP/DI/EI/HALT/STOP               ---
RETI/RETB                             old
```

Note: Opcodes that use PSW or PSW.n or CY as destination operand do of course also affect the corresponding flags.

RL78 Conditional Opcodes

The CPU supports conditional branch/skip opcodes. Skip does skip the next opcode if the condition is true (unknown if that is any faster, especially in case of large 5-byte opcodes). Combining conditional SKx+BT/BF is possible.

```
Native    Mocash
BC  SKC   JC/JB   SKC/SKB   CY=1 (carry, unsigned below)
BNC SKNC  JNC/JAE SKNC/SKAE CY=0 (no carry, unsigned above or equal)
```

```
BZ  SKZ   JZ/JE   SKZ/SKE   Z=1 (zero/equal)
BNZ SKNZ  JNZ/JNE SKNZ/SKNE Z=0 (nonzero/not equal)
BH  SKH   JA      SKA       CY=0 and Z=0 (unsigned above/higher)
BNH SKNH  JBE     SKBE      CY=1 or Z=1 (unsigned below or equal, not higher)
BF        JZ                bit=0 (bit false/zero)
BT        JNZ               bit=1 (bit true/nonzero)
BTCLR     JNZCLR            bit=1 (bit true/nonzero, and auto-clear bit)
```

There are no conditions for signed overflow/less/greater. Signed positive/negative can be tested via BF/BT on result.bit7/bit15.
The skip feature could be used as prefix for making conditional opcodes, eg. "ADD.Z" could be made of "SKNZ+ADD" (to add if zero).

Stack

```
PUSH/POP rp/PSW --> 2 bytes
CALL/CALLT/RET  --> 4 bytes
Interrupt/RETI  --> 4 bytes
BRK/RETB        --> 4 bytes
```

3DS I2C MCU - RL78 SFR Registers (Special Function Registers) (I/O ports)

Datasheets

SFRs are documented in RL78 Hardware Manuals. However, the SFRs aren't 100% same for all RL78 chips. The "78K0R/KC3-L, 78K0R/KE3-L" datasheet appears to be pretty close to the 3DS hardware (aside from the F0500h-F07FFh area).

SFR List

```
Addr   Access  Reset Symbol     Special Function Register (SFR)
FFF00h R/W 1B- 00h   P0         Port data 0
FFF01h R/W 1B- 00h   P1         Port data 1
FFF02h R/W 1B- 00h   P2         Port data 2
FFF03h R/W 1B- 00h   P3         Port data 3
FFF04h R/W 1B- 00h   P4         Port data 4
FFF05h R/W 1B- 00h   P5         Port data 5
FFF06h R/W 1B- 00h   P6         Port data 6
FFF07h R/W 1B- 00h   P7         Port data 7
FFF08h R/W 1B- 00h   P8         Port data 8   (N/A)
FFF09h R/W 1B- 00h   P9         Port data 9   (N/A)
FFF0Ah R/W 1B- 00h   P10        Port data 10  (N/A)
FFF0Bh R/W 1B- 00h   P11        Port data 11  (78K0R/K E3-L)
FFF0Ch R/W 1B- Undef P12        Port data 12
FFF0Dh R/W 1B- 00h   P13        Port data 13  (78K0R/K E3-L)
FFF0Eh R/W 1B- 00h   P14        Port data 14
FFF0Fh R/W 1B- 00h   P15        Port data 15  (N/A)
FFF10h R/W -BW 0000h SDR00      Serial data 00 (TXD0/SIO00)
FFF12h R/W -BW 0000h SDR01      Serial data 01 (RXD0/SIO01)
FFF14h R/W -BW 0000h SDR12      Serial data 12 (TXD3/SIO30)
FFF16h R/W -BW 0000h SDR13      Serial data 13 (RXD3/SIO31)
FFF18h R/W --W 0000h TDR00      Timer data 00
FFF1Ah R/W -BW 0000h TDR01(L/H) Timer data 01     (NOT L/H)
FFF1Eh R   --W 0000h ADCR       10-bit A/D conversion result
FFF1Fh R   -B- 00h   ADCRH      8-bit A/D conversion result
FFF20h R/W 1B- FFh   PM0        Port mode 0
FFF21h R/W 1B- FFh   PM1        Port mode 1
FFF22h R/W 1B- FFh   PM2        Port mode 2
FFF23h R/W 1B- FFh   PM3        Port mode 3
FFF24h R/W 1B- FFh   PM4        Port mode 4
FFF25h R/W 1B- FFh   PM5        Port mode 5
FFF26h R/W 1B- FFh   PM6        Port mode 6
```

```
 FFF27h R/W 1B- FFh   PM7        Port mode 7
 FFF28h R/W 1B- FFh   PM8        Port mode 8  (N/A)
 FFF29h R/W 1B- FFh   PM9        Port mode 9  (N/A)
 FFF2Ah R/W 1B- FFh   PM10       Port mode 10 (N/A)
 FFF2Bh R/W 1B- FFh   PM11       Port mode 11 (78K0R/K E3-L)
 FFF2Ch R/W 1B- FFh   PM12       Port mode 12
 FFF2Dh -   --- -     PM13       Port mode 13 (N/A)
 FFF2Eh R/W 1B- FFh   PM14       Port mode 14
 FFF2Fh R/W 1B- FFh   PM15       Port mode 15 (N/A) (3DS)
 FFF30h R/W 1B- 00h   ADM0       A/D converter mode 0
 FFF31h R/W 1B- 00h   ADS        Analog input channel specification
 FFF32h R/W 1B- 00h   ADM1       A/D converter mode 1                  (N/A)
 FFF37h R/W 1B- 00h   KRM        Key return mode
 FFF38h R/W 1B- 00h   EGP0       External interrupt rising edge enable 0
 FFF39h R/W 1B- 00h   EGN0       External interrupt falling edge enable 0
 FFF3Ah R/W 1B- 00h   EGP1       External int rising edge enable 1    (E3-L)
 FFF3Bh R/W 1B- 00h   EGN1       External int falling edge enable 1   (E3-L)
 FFF3Ch R/W 1B- 00h   ISC        Input switch control register (!)
 FFF3Eh R/W 1B- 00h   TIS0       Timer input select register 0 (!)
 FFF44h R/W -BW 0000h SDR02      Serial data 02 (TXD1/SIO10)
 FFF46h R/W -BW 0000h SDR03      Serial data 03 (RXD1/SIO11)
 FFF48h R/W -BW 0000h SDR10      Serial data 10 (TXD2/SIO20)  (78K0R/K E3-L)
 FFF4Ah R/W -BW 0000h SDR11      Serial data 11 (RXD2/SIO21)  (78K0R/K E3-L)
 FFF50h R/W -B- 00h   IICA0      I2C IICA shift 0
 FFF51h R   1B- 00h   IICS0      I2C IICA status 0
 FFF52h R/W 1B- 00h   IICF0      I2C IICA flag 0
;FFF54h R/W -B- 00h   IICA1      I2C IICA shift 1  (N/A)
;FFF55h R   1B- 00h   IICS1      I2C IICA status 1 (N/A)
;FFF56h R/W 1B- 00h   IICF1      I2C IICA flag 1   (N/A)
 FFF64h R/W --W 0000h TDR02      Timer data 02
 FFF66h R/W -BW 0000h TDR03(L/H) Timer data 03     (NOT L/H)
 FFF68h R/W --W 0000h TDR04      Timer data 04
 FFF6Ah R/W --W 0000h TDR05      Timer data 05
 FFF6Ch R/W --W 0000h TDR06      Timer data 06
 FFF6Eh R/W --W 0000h TDR07      Timer data 07
 FFF70h R/W --W 0000h TDR10      Timer data 10     (N/A)
 FFF72h R/W -BW 0000h TDR11(L/H) Timer data 11     (N/A)
 FFF74h R/W --W 0000h TDR12      Timer data 12     (N/A)
 FFF76h R/W -BW 0000h TDR13(L/H) Timer data 13     (N/A)
 FFF78h R/W --W 0000h TDR14      Timer data 14     (N/A)
 FFF7Ah R/W --W 0000h TDR15      Timer data 15     (N/A)
 FFF7Ch R/W --W 0000h TDR16      Timer data 16     (N/A)
 FFF7Eh R/W --W 0000h TDR17      Timer data 17     (N/A)

 FFF80h                          Undoc:
 FFF81h                          Undoc:
 FFF82h                          Undoc:
 FFF83h                          Undoc:
 FFF84h                          Undoc:
 FFF85h                          Undoc:
 FFF86h                          Undoc:
 FFF87h                          Undoc:
 FFF88h                          Undoc:
 FFF89h                          Undoc: something?
 FFF8Ah                          Undoc: something?
 FFF8Bh                          Undoc: Serial TOOL0 stat/ack
 FFF8Ch                          Undoc: Serial TOOL0 control
 FFF8Dh                          Undoc: Serial TOOL0 bauds.lsb
 FFF8Eh                          Undoc: Serial TOOL0 bauds.msb
 FFF8Fh                          Undoc: Serial TOOL0 data

 FFF90h R/W --W 0000h RSUBC      Sub-count register (!)   (3DS)  ;\either
(FFF90h)R/W --W 0FFFh ITMC       Interval timer control   (N/A)  ;/or
 FFF92h R/W -B- 00h   SEC        Second count
```

```
FFF93h R/W -B- 00h   MIN        Minute count
FFF94h R/W -B- 12h   HOUR       Hour count (12 AM upon reset)
FFF95h R/W -B- 00h   WEEK       Week count (uh, week??????)
FFF96h R/W -B- 01h   DAY        Day count
FFF97h R/W -B- 01h   MONTH      Month count
FFF98h R/W -B- 00h   YEAR       Year count
FFF99h R/W -B- 00h   SUBCUD     Watch error correction
FFF9Ah R/W -B- 00h   ALARMWM    Alarm minute
FFF9Bh R/W -B- 12h   ALARMWH    Alarm hour
FFF9Ch R/W -B- 00h   ALARMWW    Alarm week
FFF9Dh R/W 1B- 00h   RTCC0      Real-time clock control 0
FFF9Eh R/W 1B- 00h   RTCC1      Real-time clock control 1
FFF9Fh R/W 1B- 00h   RTCC2      Real-time clock control 2 (!)
FFFA0h R/W -B- 00h   CMC        Clock operation mode control
FFFA1h R/W 1B- C0h   CSC        Clock operation status control
FFFA2h R   1B- 00h   OSTC       Osc stabilization time counter status
FFFA3h R/W -B- 07h   OSTS       Osc stabilization time select
FFFA4h R/W 1B- 00h   CKC        System clock control
FFFA5h R/W 1B- 00h   CKS0       Clock output select 0
FFFA6h R/W 1B- 00h   CKS1       Clock output select 1 (N/A) (Old3DS)
FFFA8h R   -B- Undef RESF       Reset control flag
FFFA9h R/W 1B- 00h   LVIM       Low-Voltage detection
FFFAAh R/W 1B- var   LVIS       Low-Voltage detection level (reset=xxh)
FFFABh R/W -B- var   WDTE       Watchdog timer enable   (reset=1Ah/9Ah)
FFFACh R/W -B- 00h   CRCIN      CRC input            (N/A)
FFFB0h R/W -B- 00h   DSA0       DMA SFR address 0
FFFB1h R/W -B- 00h   DSA1       DMA SFR address 1
FFFB2h R/W -BW 0000h DRA0(L/H)  DMA RAM address 0
FFFB4h R/W -BW 0000h DRA1(L/H)  DMA RAM address 1
FFFB6h R/W -BW 0000h DBC0(L/H)  DMA byte count 0
FFFB8h R/W -BW 0000h DBC1(L/H)  DMA byte count 1
FFFBAh R/W 1B- 00h   DMC0       DMA mode control 0
FFFBBh R/W 1B- 00h   DMC1       DMA mode control 1
FFFBCh R/W 1B- 00h   DRC0       DMA operation control 0
FFFBDh R/W 1B- 00h   DRC1       DMA operation control 1
FFFBEh R/W 1B- 00h   BECTL      Background event control register (!)
FFFBFh     1                    Undoc:
FFFC0h -   --- Undef PFCMD      (used in the self programming library) (!)
FFFC1h     1B                   Undoc:
FFFC2h -   --- 00h   PFS        (used in the self programming library) (!)
FFFC4h -   --- 00h   FLPMC      (used in the self programming library) (!)
FFFC5h     1                    Undoc:
FFFC6h      B                   Undoc:
FFFC7h      B                   Undoc:
FFFC8h       W                  Undoc:
FFFCAh      B                   Undoc:
FFFCBh      B                   Undoc:
FFFCCh       W                  Undoc:
FFFCEh       W                  Undoc:
FFFD0h R/W 1BW 0000h IF2(L/H)   Interrupt request flag 2
FFFD2h R/W 1BW 00h   IF3(L)     Interrupt request flag 3L       (N/A)
FFFD4h R/W 1BW FFFFh MK2(L/H)   Interrupt mask flag 2
FFFD6h R/W 1BW FFh   MK3(L)     Interrupt mask flag 3L          (N/A)
FFFD8h R/W 1BW FFFFh PR02(L/H)  Priority specification flag 02
FFFDAh R/W 1BW FFh   PR03(L)    Priority specification flag 03L (N/A)
FFFDCh R/W 1BW FFFFh PR12(L/H)  Priority specification flag 12
FFFDEh R/W 1BW FFh   PR13(L)    Priority specification flag 13L (N/A)
FFFE0h R/W 1BW 0000h IF0(L/H)   Interrupt request flag 0
FFFE2h R/W 1BW 0000h IF1(L/H)   Interrupt request flag 1
FFFE4h R/W 1BW FFFFh MK0(L/H)   Interrupt mask flag 0
FFFE6h R/W 1BW FFFFh MK1(L/H)   Interrupt mask flag 1
FFFE8h R/W 1BW FFFFh PR00(L/H)  Priority specification flag 00
FFFEAh R/W 1BW FFFFh PR01(L/H)  Priority specification flag 01
FFFECh R/W 1BW FFFFh PR10(L/H)  Priority specification flag 10
FFFEEh R/W 1BW FFFFh PR11(L/H)  Priority specification flag 11
```

```
FFFF0h R/W --W 0000h MDAL       Multiplication/division data A (L) (MULA)
FFFF2h R/W --W 0000h MDAH       Multiplication/division data A (H) (MULB)
FFFF4h R/W --W 0000h MDBH       Multiplication/division data B (H) (MULOH)
FFFF6h R/W --W 0000h MDBL       Multiplication/division data B (L) (MULOL)
FFFF8h ??? ??? ????h SP(L/H)    CPU Stack Pointer                 ;\
FFFFAh R/W ??? ??h   PSW        CPU Program Status Word           ;
FFFFBh W   -B- --    -          CPU Reserve (for MUL/DIV opcodes) ; Fixed
FFFFCh ??? ??? 00h   CS         CPU Program Counter Bank          ; SFRs
FFFFDh ??? ??? ??h   ES         CPU Data Bank                     ;
FFFFEh R/W 1B- 00h   PMC        CPU Processor mode control        ;
FFFFFh ??? ??? ??h   MEM        CPU ... whatever?                 ;/
```

Extended SFR (2nd SFR) List

```
F0010h R/W 1B- 00h   ADM2      A/D converter mode 2                     (N/A)
F0011h R/W -B- FFh   ADUL      A/D result comparison upper limit setting(N/A)
F0012h R/W -B- 00h   ADLL      A/D result comparison lower limit setting(N/A)
F0013h R/W -B- 00h   ADTES     A/D test                                 (N/A)
F0017h R/W -B- 10h   ADPC      A/D port configuration register (!)
F0030h R/W 1B- 00h   PU0       Pull-up resistor option 0        ;\
F0031h R/W 1B- 00h   PU1       Pull-up resistor option 1        ;
F0032h -   --- -     PU2       Pull-up resistor option 2  (N/A) ;
F0033h R/W 1B- 00h   PU3       Pull-up resistor option 3        ;
F0034h R/W 1B- 01h   PU4       Pull-up resistor option 4        ;
F0035h R/W 1B- 00h   PU5       Pull-up resistor option 5        ;
F0036h R/W 1B- 00h   PU6       Pull-up resistor option 6  (N/A) ;
F0037h R/W 1B- 00h   PU7       Pull-up resistor option 7        ;
F0038h R/W 1B- 00h   PU8       Pull-up resistor option 8  (N/A) ;
F0039h R/W 1B- 00h   PU9       Pull-up resistor option 9  (N/A) ;
F003Ah R/W 1B- 00h   PU10      Pull-up resistor option 10 (N/A) ;
F003Bh R/W 1B- 00h   PU11      Pull-up resistor option 11 (78K0R/K E3-L)
F003Ch R/W 1B- 00h   PU12      Pull-up resistor option 12       ;
F003Dh -   --- -     PU13      Pull-up resistor option 13 (N/A) ;
F003Eh R/W 1B- 00h   PU14      Pull-up resistor option 14       ;
F003Fh -   --- -     PU15      Pull-up resistor option 15 (N/A) ;/
F0040h R/W 1B- 00h   PIM0      Port input mode 0                ;\
F0041h R/W 1B- 00h   PIM1      Port input mode 1                ;
F0042h -   --- -     PIM2      Port input mode 2  (N/A)         ;
F0043h -   --- -     PIM3      Port input mode 3  (N/A)         ;
F0044h R/W 1B- 00h   PIM4      Port input mode 4  (N/A)         ;
F0045h R/W 1B- 00h   PIM5      Port input mode 5  (N/A)         ;
F0046h -   --- -     PIM6      Port input mode 6  (N/A)         ;
F0047h -   --- -     PIM7      Port input mode 7  (N/A)         ;
F0048h R/W 1B- 00h   PIM8      Port input mode 8  (N/A)         ;
F0049h -   --- -     PIM9      Port input mode 9  (N/A)         ;
F004Ah -   --- -     PIM10     Port input mode 10 (N/A)         ;
F004Bh -   --- -     PIM11     Port input mode 11 (N/A)         ;
F004Ch -   --- -     PIM12     Port input mode 12 (N/A)         ;
F004Dh -   --- -     PIM13     Port input mode 13 (N/A)         ;
F004Eh R/W 1B- 00h   PIM14     Port input mode 14 (78K0R/K E3-L);
F004Fh -   --- -     PIM15     Port input mode 15 (N/A)         ;/
F0050h R/W 1B- 00h   POM0      Port output mode 0               ;\
F0051h R/W 1B- 00h   POM1      Port output mode 1               ;
F0052h -   --- -     POM2      Port output mode 2  (N/A)        ;
F0053h -   --- -     POM3      Port output mode 3  (N/A?) (3DS!);
F0054h R/W 1B- 00h   POM4      Port output mode 4  (N/A)        ;
F0055h R/W 1B- 00h   POM5      Port output mode 5  (N/A)        ;
F0056h -   --- -     POM6      Port output mode 6  (N/A)        ;
F0057h R/W 1B- 00h   POM7      Port output mode 7  (N/A)        ;
F0058h R/W 1B- 00h   POM8      Port output mode 8  (N/A)        ;
F0059h R/W 1B- 00h   POM9      Port output mode 9  (N/A)        ;
F005Ah -   --- -     POM10     Port output mode 10 (N/A)        ;
F005Bh -   --- -     POM11     Port output mode 11 (N/A)        ;
F005Ch -   --- -     POM12     Port output mode 12 (N/A)        ;
F005Dh -   --- -     POM13     Port output mode 13 (N/A)        ;
```

```
 F005Eh R/W 1B- 00h   POM14     Port output mode 14 (78K0R/K E3-L);
 F005Fh -   --- -     POM15     Port output mode 15 (N/A)        ;/

 F0060h R/W 1B- 00h   NFEN0     Noise filter enable 0 (!)              ;\(!)
 F0061h R/W 1B- 00h   NFEN1     Noise filter enable 1 (!)              ;/

;F0060h R/W 1B- FFh   PMC0      Port mode control 0              ;\     ;\
;F0061h -   --- -     PMC1      Port mode control 1  (N/A)       ;      ;
;F0062h -   --- -     PMC2      Port mode control 2  (N/A)       ;      ;
;F0063h R/W 1B- FFh   PMC3      Port mode control 3              ;      ;
;F0064h -   --- -     PMC4      Port mode control 4  (N/A)       ;      ;
;F0065h -   --- -     PMC5      Port mode control 5  (N/A)       ;      ;
;F0066h -   --- -     PMC6      Port mode control 6  (N/A)       ;      ;
;F0067h -   --- -     PMC7      Port mode control 7  (N/A)       ;      ; N/A
;F0068h -   --- -     PMC8      Port mode control 8  (N/A)       ;      ; !!!
;F0069h -   --- -     PMC9      Port mode control 9  (N/A)       ;      ;
;F006Ah R/W 1B- FFh   PMC10     Port mode control 10             ;      ;
;F006Bh R/W 1B- FFh   PMC11     Port mode control 11             ;      ;
;F006Ch R/W 1B- FFh   PMC12     Port mode control 12             ;      ;
;F006Dh -   --- -     PMC13     Port mode control 13 (N/A)       ;      ;
;F006Eh R/W 1B- FFh   PMC14     Port mode control 14             ;      ;
;F006Fh -   --- -     PMC15     Port mode control 15 (N/A)       ;/     ;
;F0070h R/W 1B- 00h   NFEN0     Noise filter enable 0                   ;
;F0071h R/W 1B- 00h   NFEN1     Noise filter enable 1                   ;
;F0072h R/W 1B- 00h   NFEN2     Noise filter enable 2                   ;
;F0073h R/W 1B- 00h   ISC       Input switch control                    ;
;F0074h R/W -B- 00h   TIS0      Timer input select 0                    ;
;F0076h R/W -B- 00h   ADPC      A/D port configuration                  ;
;F0077h R/W -B- 00h   PIOR      Peripheral I/O redirection              ;
;F0078h R/W -B- 00h   IAWCTL    Invalid memory access detection control ;
;F007Dh R/W 1B- 00h   GDIDIS    Global digital input disable            ;
;F0090h R/W 1B- 00h   DFLCTL    Data flash control                      ;
;F00A0h R/W -B- opt.  HIOTRM    High-speed on-chip oscillator trimming  ;
;F00A8h R/W -B- opt.  HOCODIV   High-speed on-chip oscillator frequency ;/

 F00C0h                        ?? flash?       (!)(!)(!)          ;-3DS

 F00C0h      B                  Undoc:
 F00C1h      B                  Undoc:
 F00C2h       W                 Undoc:
 F00C4h       W                 Undoc:
 F00C8h       W                 Undoc:

 F00E0h R/W --W 0000h MDCL      Multiplication/division data C (L)
 F00E2h R/W --W 0000h MDCH      Multiplication/division data C (H)
 F00E8h R/W 1B- 00h   MDUC      Multiplication/division control
 F00F0h R/W 1B- 00h   PER0      Peripheral enable 0
 F00F2h                         Timer? (!)(!)(!)                   ;-3DS
 F00F3h R/W -B- 00h   OSMC      Subsystem clock supply mode control
 F00F4h R/W -B- 00h   RMC       Regulator mode control register (!)
 F00F5h R/W 1B- 00h   RPECTL    RAM parity error control (!)
 F00F6h R/W 1B- 00h   DSCCTL    20 MHz internal high-speed osc. control (!)
 F00FEh R   -B- Undef BCDADJ    BCD adjust result
 F0100h R   -BW 0000h SSR00(L)  Serial status 00
 F0102h R   -BW 0000h SSR01(L)  Serial status 01
 F0104h R   -BW 0000h SSR02(L)  Serial status 02
 F0106h R   -BW 0000h SSR03(L)  Serial status 03
 F0108h R/W -BW 0000h SIR00(L)  Serial flag clear trigger 00
 F010Ah R/W -BW 0000h SIR01(L)  Serial flag clear trigger 01
 F010Ch R/W -BW 0000h SIR02(L)  Serial flag clear trigger 02
 F010Eh R/W -BW 0000h SIR03(L)  Serial flag clear trigger 03
 F0110h R/W --W 0020h SMR00     Serial mode 00
 F0112h R/W --W 0020h SMR01     Serial mode 01
 F0114h R/W --W 0020h SMR02     Serial mode 02
```

```
F0116h R/W --W 0020h SMR03     Serial mode 03
F0118h R/W --W 0087h SCR00     Serial communication operation setting 00
F011Ah R/W --W 0087h SCR01     Serial communication operation setting 01
F011Ch R/W --W 0087h SCR02     Serial communication operation setting 02
F011Eh R/W --W 0087h SCR03     Serial communication operation setting 03
F0120h R   1BW 0000h SE0(L)    Serial channel enable status 0
F0122h R/W 1BW 0000h SS0(L)    Serial channel start 0
F0124h R/W 1BW 0000h ST0(L)    Serial channel stop 0
F0126h R/W -BW 0000h SPS0(L)   Serial clock select 0
F0128h R/W --W 0F0Fh SO0       Serial output 0
F012Ah R/W 1BW 0000h SOE0(L)   Serial output enable 0
F0134h R/W -BW 0000h SOL0(L)   Serial output level 0
F0138h R/W -BW 0000h SSC0(L)   Serial standby control 0 (N/A)
F0140h R   -BW 0000h SSR10(L)  Serial status 10 (78K0R/K E3-L)
F0142h R   -BW 0000h SSR11(L)  Serial status 11 (78K0R/K E3-L)
F0144h R   -BW 0000h SSR12(L)  Serial status 12
F0146h R   -BW 0000h SSR13(L)  Serial status 13
F0148h R/W -BW 0000h SIR10(L)  Serial flag clear trigger 10 (78K0R/K E3-L)
F014Ah R/W -BW 0000h SIR11(L)  Serial flag clear trigger 11 (78K0R/K E3-L)
F014Ch R/W -BW 0000h SIR12(L)  Serial flag clear trigger 12
F014Eh R/W -BW 0000h SIR13(L)  Serial flag clear trigger 13
F0150h R/W --W 0020h SMR10     Serial mode 10 (78K0R/K E3-L)
F0152h R/W --W 0020h SMR11     Serial mode 11 (78K0R/K E3-L)
F0154h R/W --W 0020h SMR12     Serial mode 12
F0156h R/W --W 0020h SMR13     Serial mode 13
F0158h R/W --W 0087h SCR10     Serial communication operation setting 10 (E3)
F015Ah R/W --W 0087h SCR11     Serial communication operation setting 11 (E3)
F015Ch R/W --W 0087h SCR12     Serial communication operation setting 12
F015Eh R/W --W 0087h SCR13     Serial communication operation setting 13
F0160h R   1BW 0000h SE1(L)    Serial channel enable status 1
F0162h R/W 1BW 0000h SS1(L)    Serial channel start 1
F0164h R/W 1BW 0000h ST1(L)    Serial channel stop 1
F0166h R/W -BW 0000h SPS1(L)   Serial clock select 1
F0168h R/W --W 0F0Fh SO1       Serial output 1
F016Ah R/W 1BW 0000h SOE1(L)   Serial output enable 1
F0174h R/W -BW 0000h SOL1(L)   Serial output level 1
F0178h R/W -BW 0000h SSC1(L)   Serial standby control 1 (N/A)
...                            - - -       ?
F0180h R   --W FFFFh TCR00     Timer counter 00
F0182h R   --W FFFFh TCR01     Timer counter 01
F0184h R   --W FFFFh TCR02     Timer counter 02
F0186h R   --W FFFFh TCR03     Timer counter 03
F0188h R   --W FFFFh TCR04     Timer counter 04
F018Ah R   --W FFFFh TCR05     Timer counter 05
F018Ch R   --W FFFFh TCR06     Timer counter 06
F018Eh R   --W FFFFh TCR07     Timer counter 07
F0190h R/W --W 0000h TMR00     Timer mode 00
F0192h R/W --W 0000h TMR01     Timer mode 01
F0194h R/W --W 0000h TMR02     Timer mode 02
F0196h R/W --W 0000h TMR03     Timer mode 03
F0198h R/W --W 0000h TMR04     Timer mode 04
F019Ah R/W --W 0000h TMR05     Timer mode 05
F019Ch R/W --W 0000h TMR06     Timer mode 06
F019Eh R/W --W 0000h TMR07     Timer mode 07
F01A0h R   -BW 0000h TSR00(L)  Timer status 00
F01A2h R   -BW 0000h TSR01(L)  Timer status 01
F01A4h R   -BW 0000h TSR02(L)  Timer status 02
F01A6h R   -BW 0000h TSR03(L)  Timer status 03
F01A8h R   -BW 0000h TSR04(L)  Timer status 04
F01AAh R   -BW 0000h TSR05(L)  Timer status 05
F01ACh R   -BW 0000h TSR06(L)  Timer status 06
F01AEh R   -BW 0000h TSR07(L)  Timer status 07
F01B0h R   1BW 0000h TE0(L)    Timer channel enable status 0
F01B2h R/W 1BW 0000h TS0(L)    Timer channel start 0
F01B4h R/W 1BW 0000h TT0(L)    Timer channel stop 0
```

```
 F01B6h R/W --W 0000h TPS0      Timer clock select 0
 F01B8h R/W -BW 0000h TO0(L)    Timer output 0
 F01BAh R/W 1BW 0000h TOE0(L)   Timer output enable 0
 F01BCh R/W -BW 0000h TOL0(L)   Timer output level 0
 F01BEh R/W -BW 0000h TOM0(L)   Timer output mode 0
;F01C0h R   --W FFFFh TCR10     Timer counter 10               ;\
;F01C2h R   --W FFFFh TCR11     Timer counter 11               ;
;F01C4h R   --W FFFFh TCR12     Timer counter 12               ;
;F01C6h R   --W FFFFh TCR13     Timer counter 13               ;
;F01C8h R   --W FFFFh TCR14     Timer counter 14               ;
;F01CAh R   --W FFFFh TCR15     Timer counter 15               ;
;F01CCh R   --W FFFFh TCR16     Timer counter 16               ;
;F01CEh R   --W FFFFh TCR17     Timer counter 17               ;
;F01D0h R/W --W 0000h TMR10     Timer mode 10                  ;
;F01D2h R/W --W 0000h TMR11     Timer mode 11                  ;
;F01D4h R/W --W 0000h TMR12     Timer mode 12                  ; N/A !!!
;F01D6h R/W --W 0000h TMR13     Timer mode 13                  ;
;F01D8h R/W --W 0000h TMR14     Timer mode 14                  ;
;F01DAh R/W --W 0000h TMR15     Timer mode 15                  ;
;F01DCh R/W --W 0000h TMR16     Timer mode 16                  ;
;F01DEh R/W --W 0000h TMR17     Timer mode 17                  ;
;F01E0h R   -BW 0000h TSR10(L)  Timer status 10                ;
;F01E2h R   -BW 0000h TSR11(L)  Timer status 11                ;
;F01E4h R   -BW 0000h TSR12(L)  Timer status 12                ;
;F01E6h R   -BW 0000h TSR13(L)  Timer status 13                ;
;F01E8h R   -BW 0000h TSR14(L)  Timer status 14                ;
;F01EAh R   -BW 0000h TSR15(L)  Timer status 15                ;
;F01ECh R   -BW 0000h TSR16(L)  Timer status 16                ;
;F01EEh R   -BW 0000h TSR17(L)  Timer status 17                ;
;F01F0h R   1BW 0000h TE1(L)    Timer channel enable status 1  ;
;F01F2h R/W 1BW 0000h TS1(L)    Timer channel start 1          ;
;F01F4h R/W 1BW 0000h TT1(L)    Timer channel stop 1           ;
;F01F6h R/W --W 0000h TPS1      Timer clock select 1           ;
;F01F8h R/W -BW 0000h TO1(L)    Timer output 1                 ;
;F01FAh R/W 1BW 0000h TOE1(L)   Timer output enable 1          ;
;F01FCh R/W -BW 0000h TOL1(L)   Timer output level 1           ;
;F01FEh R/W -BW 0000h TOM1(L)   Timer output mode 1            ;
;F0200h R/W -B- 00h   DSA2      DMA SFR address 2              ;
;F0201h R/W -B- 00h   DSA3      DMA SFR address 3              ;
;F0202h R/W -BW 0000h DRA2(L/H) DMA RAM address 2              ;
;F0204h R/W -BW 0000h DRA3(L/H) DMA RAM address 3              ;
;F0206h R/W -BW 0000h DBC2(L/H) DMA byte count 2               ;
;F0208h R/W -BW 0000h DBC3(L/H) DMA byte count 3               ;
;F020Ah R/W 1B- 00h   DMC2      DMA mode control 2             ;
;F020Bh R/W 1B- 00h   DMC3      DMA mode control 3             ;
;F020Ch R/W 1B- 00h   DRC2      DMA operation control 2        ;
;F020Dh R/W 1B- 00h   DRC3      DMA operation control 3        ;/
 F0230h R/W 1B- 00h   IICCTL00  I2C IICA control 00
 F0231h R/W 1B- 00h   IICCTL01  I2C IICA control 01
 F0232h R/W -B- FFh   IICWL0    I2C IICA low-level width setting 0
 F0233h R/W -B- FFh   IICWH0    I2C IICA high-level width setting 0
 F0234h R/W -B- 00h   SVA0      I2C Slave address 0
;F0238h R/W 1B- 00h   IICCTL10  I2C IICA control 10                 (N/A)
;F0239h R/W 1B- 00h   IICCTL11  I2C IICA control 11                 (N/A)
;F023Ah R/W -B- FFh   IICWL1    I2C IICA low-level width setting 1  (N/A)
;F023Bh R/W -B- FFh   IICWH1    I2C IICA high-level width setting 1 (N/A)
;F023Ch R/W -B- 00h   SVA1      I2C Slave address 1                 (N/A)
;F02F0h R/W 1B- 00h   CRC0CTL   Flash memory CRC control          (N/A)
;F02F2h R/W --W 0000h PGCRCL    Flash memory CRC operation result (N/A)
;F02FAh R/W --W 0000h CRCD      CRC data                          (N/A)
For 78K0R...
 F0540h..   ..  ..    UF0..     UF0 stuff.. etc.. (not present in 3DS)
Below are 3DS specific...
 F0501h     ?   ?              ??   Peripheral enable 1 for I2C ?  ;-3DS
 F0510h     ?   ?              Port data X ?                       ;\
```

```
F0511h     ?   ?              Port mode X                          ; 3DS
F0512h     ?   ?              Pull-up resistor option X            ;/
F0538h     ?   ?              ??                                   ;-3DS
F0540h R/W -B- 00h   IICA1     I2C IICA shift 1                     ;\
F0541h R   1B- 00h   IICS1     I2C IICA status 1                    ; 3DS
F0542h R/W 1B- 00h   IICF1     I2C IICA flag 1                      ;/
F0550h R/W 1B- 00h   IICCTL10  I2C IICA control 10                  ;\
F0551h R/W 1B- 00h   IICCTL11  I2C IICA control 11                  ;
F0552h R/W -B- FFh   IICWL1    I2C IICA low-level width setting 1   ; 3DS
F0553h R/W -B- FFh   IICWH1    I2C IICA high-level width setting 1  ;
F0554h R/W -B- 00h   SVA1      I2C Slave address 1                  ;/

F0746h      B                  Undoc:                                    ;-??
F07E0h       W                 Undoc: Saved AX               ;\          ;\
F07E2h       W                 Undoc: Saved BC               ;           ;
F07E4h       W                 Undoc: Saved DE               ; debug     ; RAM
F07E6h       W                 Undoc: Saved HL               ; stack?    ;
F07E8h       W                 Undoc: Saved pop (maybe PSW?) ;           ;
F07EAh      B                  Undoc: Saved CS               ;           ;
F07EBh      B                  Undoc: Saved ES               ;/          ;
F07ECh      B                  Undoc: Security okay flag (55h=yes)       ;
F07EDh      B                  Undoc: Saved A ?                          ;
F07EEh      B                  Undoc: Reply value ?                      ;
F07F0h      B                  Undoc: RET opcode (or custom vector)      ;
                                                                         ;
F0800h R/W  B  ?               FLASH related                        ;\   ;
F0801h   W  B  ?               FLASH related                        ;    ;
F0802h   W  B  ?               FLASH related                        ; 3DS;
F0803h   W  B  ?               FLASH related                        ;    ;
F0804h R/W  BW ?               FLASH related address?               ;    ;
F0806h R/W  B  ?               FLASH related                        ;/   ;/
```

CPU Register Banks & short-addressable RAM

```
FFE20h R/W 1BW ??    RAM[C0h]  (can be accessed with "short" address)
FFEE0h R/W 1BW ??    AX3(X3/A3) AX Bank 3 aka RP0(3) aka R0(3)/R1(3) ;\
FFEE2h R/W 1BW ??    BC3(C3/B3) BC Bank 3 aka RP1(3) aka R2(3)/R3(3) ; Bank 3
FFEE4h R/W 1BW ??    DE3(E3/D3) DE Bank 3 aka RP2(3) aka R4(3)/R5(3) ;
FFEE6h R/W 1BW ??    HL3(L3/H3) HL Bank 3 aka RP3(3) aka R6(3)/R7(3) ;/
FFEE8h R/W 1BW ??    AX2(X2/A2) AX Bank 2 aka RP0(2) aka R0(2)/R1(2) ;\
FFEEAh R/W 1BW ??    BC2(C2/B2) BC Bank 2 aka RP1(2) aka R2(2)/R3(2) ; Bank 2
FFEECh R/W 1BW ??    DE2(E2/D2) DE Bank 2 aka RP2(2) aka R4(2)/R5(2) ;
FFEEEh R/W 1BW ??    HL2(L2/H2) HL Bank 2 aka RP3(2) aka R6(2)/R7(2) ;/
FFEF0h R/W 1BW ??    AX1(X1/A1) AX Bank 1 aka RP0(1) aka R0(1)/R1(1) ;\
FFEF2h R/W 1BW ??    BC1(C1/B1) BC Bank 1 aka RP1(1) aka R2(1)/R3(1) ; Bank 1
FFEF4h R/W 1BW ??    DE1(E1/D1) DE Bank 1 aka RP2(1) aka R4(1)/R5(1) ;
FFEF6h R/W 1BW ??    HL1(L1/H1) HL Bank 1 aka RP3(1) aka R6(1)/R7(1) ;/
FFEF8h R/W 1BW ??    AX0(X0/A0) AX Bank 0 aka RP0(0) aka R0(0)/R1(0) ;\
FFEFAh R/W 1BW ??    BC0(C0/B0) BC Bank 0 aka RP1(0) aka R2(0)/R3(0) ; Bank 0
FFEFCh R/W 1BW ??    DE0(E0/D0) DE Bank 0 aka RP2(0) aka R4(0)/R5(0) ;
FFEFEh R/W 1BW ??    HL0(L0/H0) HL Bank 0 aka RP3(0) aka R6(0)/R7(0) ;/
```

The "1BW" column indicates support for Bit/Byte/Word access.
Words with (L) or (L/H) symbol suffix can be alternately accessed as low and/or high byte.

3DS I2C MCU - RL78 Misc

Overlapping

```
ADDW AX,saddrp   can also be used for first some SFR's FFF00..FFF1F
CLRB saddr       can also be used for first some SFR's FFF00..FFF1F
addr16           can be used for further SFRs          FFF20..FFFFF
SFR_SP/PSW/ES   can be used as alias for SP/PSW/ES
```

```
SFR_AX/BC/DE/HL can be used as alias for AX/BC/DE/HL - but banked!
SFR_CS    ... can be manually changed BEFORE JMP/CALL AX to make FAR-JMP... ?
```

Nocash Syntax

```
Native                       Nocash
CLRB/CLRW/CMP0 op            MOV/MOVW/CMP op,0
ONEB/ONEW op                 MOV/MOVW op,1
ADDC/SUBC/ROLC/RORC/ROLWC    ADC/SBC/RCL/RCR/RCRW/RCLW
BR/BF/BT/Bcond/BTCLR         JMP/JZ/JNZ/Jcond/JNZCLR
CALLT                        CALL
MULU X                       MULU AX,A,X    ;optional alias

@@def Vbyte            #byte
@@def Vword            #word
@@def sfr              sfr        SFR name aka FFF00h..FFFFFh
@@def sfrp             sfrp       16-bit-manipulatable SFR name (even address)
@@def saddr            saddr      8bit data addr at FFE20h to FFF1Fh
@@def saddrp           saddrp     same as saddr, but even-address only
@@def $addr20          $addr20    probably 8bit reljump
@@def $Xaddr20         $!addr20   probably 16bit reljump
@@def XXaddr20_code    !!addr20   20bit jumpdest (3 bytes) (CALL/BR) to CS=x
@@def Xaddr16_code     !addr16    16bit jumpdest (2 bytes) (CALL/BR) to CS=0!
@@def Xaddr16          !addr16    16bit data addr                    at ES=F
@@def IhlI             [HL]           ;[HL]        ;\
@@def IdeI             [DE]           ;[DE]        ; maybe can also use ES:?
@@def Ihl_bI           [HL+B]         ;[HL+B]      ;
@@def Ihl_cI           [HL+C]         ;[HL+C]      ;/
@@def wordIbcI         word[BC]       ;[BC+nnnn]   ;\
@@def wordIbI          word[B]        ;[B+nnnn]    ;
@@def wordIcI          word[C]        ;[C+nnnn]    ; can use ES:
@@def Ihl_byteI        [HL+byte]      ;[HL+nn]     ;
@@def Ide_byteI        [DE+byte]      ;[DE+nn]     ;/
@@def Isp_byteI        [SP+byte]      ;[SP+nn]     ;\shouldn't/can't use ES:?
@@def I00nnI           [00nn]         ;[000nn]     ;/
```

Caution: 16bit MOVW/ADDW/etc with memory operand must use even addresses (this is documented only for [sfrp] and [saddrp], but does also apply for [nnnn], which does actually read/write memory at [nnnn and FFFEh]).

Identifier Description

```
?  [HL + byte], [DE + byte], [SP + byte] (only the space from F0000H
    to FFFFFH is specifiable)
?  word[B], word[C] (only the space from F0000H to FFFFFH is specifiable)
?  word[BC] (only the space from F0000H to FFFFFH is specifiable)
?  ES:[HL + byte], ES:[DE + byte] (higher 4-bit addresses are specified by
    the ES register)
?  ES:word[B], ES:word[C] (higher 4-bit addresses are specified by
    the ES register)
?  ES:word[BC] (higher 4-bit addresses are specified by the ES register)
```

MCU-FIRM

```
00001F02 MCU    v0, v1026, v2048, v3072, v4102, v5122, v6145, v7168, v8192
20001F02 New_3DS MCU            v8192, v9216(New2DSXL)
00001F03 SAFE_MODE MCU          v0
20001F03 New_3DS SAFE_MODE MCU  v9217
```

RL78/G13 - Memory Map (R5F100xA, R5F101xA (x = 6 to 8, A to C, E to G))

```
00000h Boot Cluster 0
01000h Boot Cluster 1     ;<-- hence the gap
```

3DS I2C Gyroscope (old version)

CAUTION: This is for a reportedly existing gyroscope that doesn't actually exist, at least not in New3DS. New3DS returns only FFh's when trying to read below registers.

GYRO Register Map - InvenSense ITG-3200 gyroscope

```
  Hex     Register Name R/W
  00h     WHO_AM_I      R/W Device ID (aka Who Am I)
  15h     SMPLRT_DIV    R/W Sample Rate Divider (reset=00h)
  16h     DLPF_FS       R/W Low Pass Filter and Full Scale Config (reset=00h)
  17h     INT_CFG       R/W Interrupt Configuration (reset=00h)
  1Ah     INT_STATUS    R   Interrupt Status (reset=00h)
  1Bh/1Ch TEMP_OUT_H/L  R   signed 16bit Temperature data (R)
  1Dh/1Eh GYRO_XOUT_H/L R   signed 16bit Gyro X output data (R)
  1Fh/20h GYRO_YOUT_H/L R   signed 16bit Gyro Y output data (R)
  21h/22h GYRO_ZOUT_H/L R   signed 16bit Gyro Y output data (R)
  3Eh     PWR_MGM       R/W Power Management (reset=00h)
  xxh-FFh Undocumented (unknown)
```

Note: The 16bit H/L values are in BIG-ENDIAN format.

Register Description

The register space allows single-byte reads and writes, as well as burst reads and writes. When performing burst reads or writes, the memory pointer will increment until reading or writing is terminated by the master, or until the memory pointer reaches "certain reserved registers between 21h and 3Ch" (uh, that would include GYRO_ZOUT at 21h and/or 22h?).

Note that any bit that is not defined should be set to zero.

GYRO[00h] - WHO_AM_I - Device ID (R/W)

```
  0    Reserved ("may be 0 or 1") (maybe ID.bit1, maybe R/W, or what?)    (?)
  1-6  I2C Device ID bit2-7 (initially 34h on power-up, aka D0h/4)        (R/W)
  7    Reserved ("should be 0")                                           (?)
```

Contains the upper 6bit of the 8bit I2C Device ID, can be used for detection, the value is also write-able and can be changed after power-on.

GYRO[15h] - SMPLRT_DIV - Sample Rate Divider (R/W)

```
  0-7  Sample Rate Divider (00h..FFh=Divide by 1..100h) (initially 00h)
```

The gyros outputs are sampled internally at either 1kHz or 8kHz, determined by the DLPF_CFG setting (see register 22). This sampling is then filtered digitally and delivered into the sensor registers after the number of cycles determined by this register. The sample rate is given by the following formula:

```
  Fsample = Finternal / (divider+1)   ;where Finternal is either 1kHz or 8kHz
```

As an example, if the internal sampling is at 1kHz, then setting this register to 7 would give the following:

```
  Fsample = 1kHz / (7 + 1) = 125Hz, or 8ms per sample
```

GYRO[16h] - DLPF_FS - Low Pass Filter and Full Scale Config (R/W)

```
  0-2  DLPF_CFG  Digital low pass filter bandwidth & internal sample rate
  3-4  FS_SEL    Full scale selection for gyro sensor data
  5-7  Reserved (should be 0)
```

FS_SEL Gyro Full-Scale Range:

```
  0 Reserved (despite of being power-up default)
  1 Reserved
  2 Reserved
  3 +/-2000ø/sec (this should be used)
```

DLPF_CFG, Low Pass Filter Bandwidth, Internal Sample Rate:

```
  0 Internal Sample Rate=8kHz, Low Pass Filter Bandwidth=256Hz
  1 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=188Hz
  2 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=98Hz
  3 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=42Hz
  4 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=20Hz
  5 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=10Hz
```

```
  6 Internal Sample Rate=1kHz, Low Pass Filter Bandwidth=5Hz
  7 Reserved
```

GYRO[17h] - Interrupt Enable and INT Pin Configuration (R/W)

```
  0  Enable interrupt when new sample data is available           (1=Enable)
  1  -
  2  Enable interrupt when PLL ready ;after changing clock source (1=Enable)
  3  -
  4  Latch clear method (0=Status register read only, 1=Any register read);\
  5  Latch mode         (0=50us pulse, 1=latch until interrupt is cleared); INT
  6  Drive type for INT output pin  (0=push-pull, 1=open drain)           ; pin
  7  Logic level for INT output pin (0=active high, 1=active low)         ;/
```

Unknown what the PLL ready IRQ is good for, maybe the whole I2C bus becomes unstable/inactive and shouldn't be used until receiving the IRQ?

GYRO[1Ah] - Interrupt Status (R)

```
  0   New sample data is ready          (0=No, 1=Yes/IRQ)
  1   -
  2   PLL clock source change ready     (0=No, 1=Yes/IRQ)
  3-7 -
```

If the interrupt is not enabled, the associated status bit will NOT get set!
Interrupt Status bits are cleared after reading GYRO[1Ah] (or optionally, if GYRO[17h].bit4=1, after reading any GYRO[xxh] register).

GYRO[1Bh/1Ch] - TEMP_OUT_H/L signed 16bit Temperature data (R)
GYRO[1Dh/1Eh] - GYRO_XOUT_H/L signed 16bit Gyro X output data (R)
GYRO[1Fh/20h] - GYRO_YOUT_H/L signed 16bit Gyro Y output data (R)
GYRO[21h/22h] - GYRO_ZOUT_H/L signed 16bit Gyro Y output data (R)
Sensor Registers. These values can be read at any time; however it is best to use the interrupt function to determine when new data is available.

GYRO[3Eh] - Power Management (R/W)

```
  0-2 CLK_SEL  Select device clock source  (see below)
  3   STBY_ZG  Gyro Z standby mode  (0=Normal, 1=Standby)  ;\eg. disable
  4   STBY_YG  Gyro Y standby mode  (0=Normal, 1=Standby)  ; unused axis
  5   STBY_XG  Gyro X standby mode  (0=Normal, 1=Standby)  ;/to save power
  6   SLEEP    Enable low power sleep mode (0=Normal, 1=Very low power sleep)
  7   H_RESET  Reset device and internal registers to power-up-default settings
```

The CLK_SEL setting determines the device clock source as follows:

```
  0 Internal oscillator (default on power-up, but unreliable, PLLs are better)
  1 PLL with X Gyro reference
  2 PLL with Y Gyro reference
  3 PLL with Z Gyro reference
  4 PLL with external 32.768kHz reference  ;\unknown if available in 3DS,
  5 PLL with external 19.2MHz reference    ;/maybe from 32KHz RTC output?
  6 Reserved
  7 Reserved
```

Gyro Datasheets

```
  Hardware: ITG-3200-Datasheet.pdf    ITG-3200 Product Specification (v1.7)
  Software: ITG-3200-Register-Map.pdf ITG-3200 Register Map (v1.0)
```

Older versions did include the Register Map in the Product Specification.

3DS I2C Gyroscope (new version)

NEW_GYRO Register Map - InvenSense ITG-1010

```
  04h/05h XG_OFFS_TC_H/L R/W Temperature Compensation X (10bit)
  07h/08h YG_OFFS_TC_H/L R/W Temperature Compensation Y (10bit)
```

```
  0Ah/0Bh ZG_OFFS_TC_H/L R/W Temperature Compensation Z (10bit)
  13h/14h XG_OFFS_USRH/L R/W User DC Bias Compensation X (16bit)
  15h/16h YG_OFFS_USRH/L R/W User DC Bias Compensation Y (16bit)
  17h/18h ZG_OFFS_USRH/L R/W User DC Bias Compensation Z (16bit)
  19h     SMPLRT_DIV     R/W Sample Rate Divider
  1Ah     CONFIG         R/W Configuration
  1Bh     GYRO_CONFIG    R/W Gyroscope Configuration
  23h     FIFO_EN        R/W FIFO Channel Enable
  37h     INT_PIN_CFG    R/W INT Pin / Bypass Enable Configuration
  38h     INT_ENABLE     R/W Interrupt Enable
  3Ah     INT_STATUS     R   Interrupt Status
  41h/42h TEMP_OUT_H/L   R   Temperature Measurement  TEMP_OUT  (16bit)
  43h/44h GYRO_XOUT_H/L  R   Gyroscope Measurements X GYRO_XOUT (16bit)
  45h/46h GYRO_YOUT_H/L  R   Gyroscope Measurements Y GYRO_YOUT (16bit)
  47h/48h GYRO_ZOUT_H/L  R   Gyroscope Measurements Z GYRO_ZOUT (16bit)
  6Ah     USER_CTRL      R/W User Control
  6Bh     PWR_MGMT_1     R/W Power Management 1
  6Ch     PWR_MGMT_2     R/W Power Management 2
  72h/73h FIFO_COUNTH/L  R/W FIFO Count Register (10bit)
  74h     FIFO_R_W       R/W FIFO Read/Write Data
  75h     WHO_AM_I       R   Device ID       - WHO_AM_I[6:1] -
  80h-FFh Undocumented (in New3DSXL these are same as 00h-7Fh)
```
Note: The 16bit H/L values are in BIG-ENDIAN format.

NEW_GYRO[04h/05h] - XG_OFFS_TC_H/L - Temperature Compensation X (10bit, R/W)
NEW_GYRO[07h/08h] - YG_OFFS_TC_H/L - Temperature Compensation Y (10bit, R/W)
NEW_GYRO[0Ah/0Bh] - ZG_OFFS_TC_H/L - Temperature Compensation Z (10bit, R/W)
Unknown if these registers are implemented in 3DS (register 00h..0Bh contain "garbage" that rather doesn't look like 10bit values).
```
  0-9   Signed 10bit, in 2.52 mdps/C steps (default=factory programmed)
  10-15 Unused (should be 0)
```
Writing 0000h disables temperature compensation (this doesn't affect the factory values, which will be restored upon reset/power-up).

NEW_GYRO[13h/14h] - XG_OFFS_USRH/L - User DC Bias Compensation X (16bit, R/W)
NEW_GYRO[15h/16h] - YG_OFFS_USRH/L - User DC Bias Compensation Y (16bit, R/W)
NEW_GYRO[17h/18h] - ZG_OFFS_USRH/L - User DC Bias Compensation Z (16bit, R/W)
```
  0-15  Signed 16bit, subtracted from the sensor values
```

NEW_GYRO[19h] - SMPLRT_DIV - Sample Rate Divider (R/W)
```
  0-7  Sample Rate Divider (00h..FFh=Divide by 1..100h) (initially 00h)
```
This register specifies the divider from the gyroscope output rate that can be used to generate a reduced Sample Rate.
```
  When FCHOICE_B=0 and DLPF_CFG=1..6   --> Sample Rate Divider IS USED
  When FCHOICE_B=0 and DLPF_CFG=0 or 7 --> Sample Rate is fixed at 8kHz ???
  When FCHOICE_B=1..3                  --> Sample Rate is fixed at 32kHz
```
The sensor register output and FIFO output are both based on the Sample Rate.
When this register is effective under the FCOICE_B and DLPF_CFG settings, the reduced Sample Rate is generated by the formula below:
```
  Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)
```
where Gyroscope Output Rate = 1kHz.

NEW_GYRO[1Ah] - CONFIG - Configuration (R/W)
```
  0-2 DLPF_CFG      Configures the Digital Low Pass Filter DLPF setting
  3-5 EXT_SYNC_SET  Configures the Frame Sync FSYNC pin sampling
  6   FIFO_MODE     FIFO Mode (0=Overwrite oldest data, 1=Stop when full)
  7   -             Reserved
```
Please note that the DLPF can only be used when FCHOICE_B=0 (Register 27).
An external signal connected to the FSYNC pin can be sampled by configuring EXT_SYNC_SET.
Signal changes to the FSYNC pin are latched so that short strobes may be captured. The latched FSYNC signal

will be sampled at the Sampling Rate, as defined in register 25. After sampling, the latch will reset to the current FSYNC signal state.
The sampled value will be reported in place of the least significant bit in a sensor data register determined by the value of EXT_SYNC_SET according to the following table.

EXT_SYNC_SET	FSYNC Bit Location
0	Input disabled
1	TEMP_OUT_L.bit0
2	GYRO_XOUT_L.bit0
3	GYRO_YOUT_L.bit0
4	GYRO_ZOUT_L.bit0

The DLPF is configured by DLPF_CFG, when FCHOICE_B=0. The gyroscope and temperature sensor are filtered according to the value of DLPF_CFG and FCHOICE_B as shown in the table below.

FCHOICE_B		DLPF_CFG	<---Gyroscope------->			<--Temperature Sensor-->	
<1>	<0>		Bandwidth	Delay	Fs	Bandwidth	Delay
0	0	0	250Hz	0.97ms	8kHz	4000Hz	0.04ms
0	0	1	184Hz	2.9ms	1kHz	188Hz	1.9ms
0	0	2	92Hz	3.9ms	1kHz	98Hz	2.8ms
0	0	3	41Hz	5.9ms	1kHz	42Hz	4.8ms
0	0	4	20Hz	9.9ms	1kHz	20Hz	8.3ms
0	0	5	10Hz	17.85ms	1kHz	10Hz	13.4ms
0	0	6	5Hz	33.48ms	1kHz	5Hz	18.6ms
0	0	7	3600Hz	0.17ms	8kHz	4000Hz	0.04ms
-	1	-	8800Hz	0.064ms	32kHz	4000Hz	0.04ms
1	0	-	3600Hz	0.11ms	32kHz	4000Hz	0.04ms

NEW_GYRO[1Bh] - GYRO_CONFIG - Gyroscope Configuration (R/W)

0-1	FCHOICE_B	Choose the gyroscope output setting
2	-	Reserved
3-4	FS_SEL	Gyroscope Full Scale range select
5	ZG_ST	Gyroscope X self test (0=Normal, 1=Perform test)
6	YG_ST	Gyroscope Y self test (0=Normal, 1=Perform test)
7	XG_ST	Gyroscope Z self test (0=Normal, 1=Perform test)

The electro/mechanical self test bits will move the sensor's proof masses, the result can be then seen in sensor registers [43h..48h].

FS_SEL	Full Scale Range
0	+/-250 ø/s
1	+/-500 ø/s
2	+/-1000 ø/s
3	+/-2000 ø/s

FCHOICE_B, in conjunction with DLPF_CFG (Register 1Ah), is used to choose the gyroscope output setting. For further information regarding the operation of FCHOICE_B, please refer to Section 4.2.

NEW_GYRO[23h] - FIFO_EN - R/W FIFO Channel Enable (R/W)

0-3	-	Reserved
4	ZG_FIFO_EN	Output Gyroscope Z to FIFO (0=No, 1=Yes)
5	YG_FIFO_EN	Output Gyroscope Y to FIFO (0=No, 1=Yes)
6	XG_FIFO_EN	Output Gyroscope X to FIFO (0=No, 1=Yes)
7	TEMP_FIFO_EN	Output Temperature to FIFO (0=No, 1=Yes)

NEW_GYRO[37h] - INT_PIN_CFG - INT Pin / Bypass Enable Configuration (R/W)

0-1	-	Reserved
2	FSYNC_INT_MODE_EN	FSYNC Pin Enable (0=Disable, 1=Enable FSYNC IRQ)
3	FSYNC_INT_LEVEL	FSYNC Pin Act Low (0=Active High, 1=Active Low)
4	INT_RD_CLEAR	Interrupt Clear (0=Upon IntStatus read, 1=Upon Any read)
5	LATCH_INT_EN	INT Pin Hold (0=50us pulse, 1=Hold until ack)
6	INT_OPEN	INT Pin Open Drain (0=Push-pull, 1=Open drain)
7	INT_LEVEL	INT Pin Active Low (0=Active High, 1=Active Low)

FSYNC_INT_MODE_EN: When bit2 is equal to 1, the FSYNC pin will trigger an interrupt when it transitions to the level specified by FSYNC_INT_LEVEL. When a FSYNC interrupt is triggered, the FSYNC_INT bit in Register 58 will be set to 1. An interrupt is sent to the host processor if the FSYNC interrupt is enabled by the FSYNC_INT_EN bit in Register 56.

NEW_GYRO[38h] - INT_ENABLE - Interrupt Enable (R/W)

```
  0   DATA_RDY_EN   Enable Data Ready interrupt          (0=Disable, 1=Enable)
  1-2 -             Reserved
  3   FSYNC_INT_EN  Enable FSYNC pin interrupt (0=Disable, 1=Use FSYNC as IRQ)
  4   FIFO_OFLOW_EN Enable FIFO buffer overflow interrupt (0=Disable, 1=Enable)
  5-7 -             Reserved
```

The Data Ready interrupt is triggered when all the sensor registers have been written with the latest gyro sensor data.

NEW_GYRO[3Ah] - INT_STATUS - Interrupt Status (R)

```
  0   DATA_RDY_INT   Data Ready (sample rate) interrupt (0=No, 1=Yes/IRQ)
  1-2 -              Reserved
  3   FSYNC_INT      FSYNC Frame Sync interrupt         (0=No, 1=Yes/IRQ)
  4   FIFO_OFLOW_INT FIFO Overrun interrupt             (0=No, 1=Yes/IRQ)
  5-7 -              Reserved
```

Each bit will clear after the register is read.

NEW_GYRO[41h/42h] - TEMP_OUT_H/L - Temperature Measurement (R, 16bit)
NEW_GYRO[43h/44h] - GYRO_XOUT_H/L - Gyroscope Measurements X (R, 16bit)
NEW_GYRO[45h/46h] - GYRO_YOUT_H/L - Gyroscope Measurements Y (R, 16bit)
NEW_GYRO[47h/48h] - GYRO_ZOUT_H/L - Gyroscope Measurements Z (R, 16bit)

```
  0-15  Most recent sensor value, signed 16bit, updated at sample rate
```

These sensor registers [41h..48h], are composed of two sets of registers: an internal register set and a user-facing read register set.

```
 The internal register set is always updated at the Sample Rate.
 The user-facing read register set duplicates the internal register set's
 data values whenever the serial interface is idle. This guarantees that a
 burst read of sensor registers will read measurements from the same sampling
 instant.
```

Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.
Temperature: The scale factor and offset for the temperature sensor are found in the Electrical Specifications table in Product Specification document.

NEW_GYRO[6Ah] - USER_CTRL - User Control (R/W)

```
  0   Reset Gyro/Temp sensor signal paths (0=No/ready, 1=Reset/busy?)
  1   Reserved
  2   FIFO Reset (clear fifo)            (0=No/ready, 1=Reset/busy?)
  3   Reserved
  5   Reserved
  4   Primary Interface select           (0=I2C/normal, 1=SPI/instead)
  6   FIFO Read Enable                   (0=Disable, 1=Enable)
  7   Reserved
```

It is "recommended" to have FIFO Read disabled when doing FIFO Reset.

NEW_GYRO[6Bh] - PWR_MGMT_1 - Power Management 1 (R/W)

```
  0-2 CLKSEL       Clock source select (see below)
  3   TEMP_DIS     Temperature sensor disable   (0=Normal, 1=Disable)
  4-5 -            Reserved
  6   SLEEP        Sleep Mode                   (0=Normal, 1=Sleep)
  7   DEVICE_RESET Reset all internal registers (0=No/ready, 1=Reset/busy?)
```

CLKSEL Clock Source

```
  00h = Internal 20MHz oscillator
  01h = PLL  ;\
  02h = PLL  ; gyroscope based clock (PLL)
  03h = PLL  ;
  04h = PLL  ;
  05h = PLL  ;/
```

```
  06h = Internal 20MHz oscillator
  07h = Reserved
```
The PLL is the default clock source upon power up (uh, but all registers except WHO_AM_I are said to be zero on power-up). In order for the gyroscope to perform to spec, the PLL must be selected as the clock source.
When the internal 20MHz oscillator is chosen as the clock source, the device can operate while having the gyroscopes disabled. However, this is only recommended if the user wishes to use the internal temperature sensor in this mode.
For further information regarding the device clock source, please refer to the relevant "Product Specification document" (uh, but this sentence comes from the Product Specification) and the Power Mode Transition Descriptions section in the Appendix (uh, but there is no Appendix).
Reset: The default values for each register can be found in "Section 3", uh?

NEW_GYRO[6Ch] - PWR_MGMT_2 - Power Management 2 (R/W)
```
  0   STBY_ZG Gyro Z standby mode  (0=Normal, 1=Standby)
  1   STBY_YG Gyro Y standby mode  (0=Normal, 1=Standby)
  2   STBY_XG Gyro X standby mode  (0=Normal, 1=Standby)
  3-5 -       Reserved bit(s)
  6-7 -       Reserved bit-pair?
```
To activate any gyro axis again, all three gyro axes must first be put into standby mode, and then be turned on simultaneously.
If the user wishes to put all three gyro axes into standby mode, the internal oscillator must be selected as the clock source.
If all three gyro axes are put into standby mode while the clock source of the device is set to the PLL (with the gyro drive generating the reference clock), the chip will hang due to an absence of a clock.
As long as one gyro axis is enabled, the drive circuit will remain active and the PLL will provide a clock.

NEW_GYRO[72h/73h] - FIFO_COUNTH/L - FIFO Count Register (10bit) (R/W)
```
  0-9   Number of bytes stored in the FIFO buffer (max=?)
  10-15 Reserved
```
The count value is latched when reading the High byte (ie. one can read H-then-L without risking carry-outs from L to H during reading).

NEW_GYRO[74h] - FIFO_R_W - FIFO Read/Write Data (R/W)
```
  0-7   FIFO Data
```
This register is used to read and write(?) data from the FIFO buffer.
Data is written to the FIFO in order of register number, (from lowest to highest). If all the FIFO enable flags (see below) are enabled, the contents of NEW_GYRO[41h..48h] will be written in order.

NEW_GYRO[75h] - WHO_AM_I - Device ID (always 68h) (R)
```
  0   Reserved ("Hard coded to 0")                              (R)
  1-6 I2C Device ID bit2-7 (always 34h, aka D0h/4 or D2h/4)     (R)
  7   Reserved ("Hard coded to 0")                              (R)
```
The device address D0h or D2h can be selected via AD0 pin (the AD0 pin is not reflected in this register). This register is read-only (unlike as OLD_GYRO).

NEW_GYRO[00h..0Dh] - Reserved/Undocumented
Reading these fourteen registers returns following weird values:
```
  BD DB D1 29 9C 1D 20 67 F4 1A 8C 08 B1 71
```
There should be 10bit temperature calibration values, but 9C1Dh,67F4h,8C08h exceed 10bit range.
The first two bytes (BDh,DBh) seem to resemble the 1st/2nd bytes of older RS232 gyroscope protocols.

3DS I2C Infrared Receiver/Transmitter (IrDA)

NXP SC16IS750
Single UART with I2C-bus/SPI interface, 64 bytes of transmit and receive FIFOs, IrDA SIR built-in support -

Register address byte (I2C)

```
  7    Unused (0)
  6-3  UART's internal register select
  2-1  Channel select (0) ;other values are reserved and should not be used.
  0    Unused (0)
```

Register set

```
 Normal register set (default mapping)
  00h.DLAB=0.R     RHR       Receive Holding Register
  00h.DLAB=0.W     THR       Transmit Holding Register
  08h.DLAB=0.R/W   IER       Interrupt Enable Register
  10h.W            FCR       FIFO Control Register
  10h.R            IIR       Interrupt Identification Register
  18h.R/W          LCR       Line Control Register
  20h.R/W          MCR       Modem Control Register
  28h.R            LSR       Line Status Register
  30h.normal.R     MSR       Modem Status Register
  38h.normal.R/W   SPR    Scratchpad Register
  40h.R            TXLVL     Transmitter FIFO Level
  48h.R            RXLVL     Receiver FIFO Level
  50h.R/W          IODir     Programmable I/O pins Direction  ;\IO only on
  58h.R/W          IOState   Programmable I/O pins State      ; SC16IS750
  60h.R/W          IOIntEna  I/O Interrupt Enable Register    ; and
  68h              reserved  Reserved (00h)                   ; SC16IS760
  70h.R/W          IOControl I/O Control register             ;/
  78h.R/W          EFCR      Extra Features Control Register
 Special/Enhanced registers (mapped depending on LCR/MCR/EFR bits):
  00h.DLAB=1.R/W   DLL       Baudrate Divisor Latch Low    ;\when LCR.bit7=1
  08h.DLAB=1.R/W   DLH       Baudrate Divisor Latch High   ;/and LCR<>BFh
  10h.LCR=BFh.R/W  EFR       Enhanced Features Register              ;\
  20h.LCR=BFh.R/W  XON1   Xon1 word                                  ; when
  28h.LCR=BFh.R/W  XON2   Xon2 word                                  ; LCR=BFh
  30h.LCR=BFh.R/W  XOFF1  Xoff1 word (whatever, maybe OUTGOING char?) ;
  38h.LCR=BFh.R/W  XOFF2  Xoff2 word (compare for INCOMING chars)    ;/
  30h.special.R/W  TCR       Transmission Control Register ;\when MCR.bit2=1
  38h.special.R/W  TLR       Trigger Level Register        ;/and EFR.bit4=1
```

[2] These bits in can only be modified if register bit EFR[4] is enabled.
[5] After Receive FIFO or Transmit FIFO reset (through FCR[1:0]), the user must wait at least 2' ??? Tclk of XTAL1 before reading or writing data to RHR and THR, respectively.
[8] IrDA mode slow/fast for SC16IS760, slow only for SC16IS750.

IR[00h.DLAB=0.R] - RHR - Receive Holding Register (R)

The receiver section consists of the Receiver Holding Register (RHR) and the Receiver Shift Register (RSR). The RHR is actually a 64-byte FIFO. The RSR receives serial data from the RX pin. The data is converted to parallel data and moved to the RHR. The receiver section is controlled by the Line Control Register. If the FIFO is disabled, location zero of the FIFO is used to store the characters.

IR[00h.DLAB=0.W] - THR - Transmit Holding Register (W)

The transmitter section consists of the Transmit Holding Register (THR) and the Transmit Shift Register (TSR). The THR is actually a 64-byte FIFO. The THR receives data and shifts it into the TSR, where it is converted to serial data and moved out on the TX pin. If the FIFO is disabled, the FIFO is still used to store the byte. Characters are lost if overflow occurs.

IR[10h.W] - FCR - FIFO Control Register (W)

```
  7-6  Trigger level for the RX FIFO (0..3 = 8,16,56,60 chars)
  5-4  Trigger level for the TX FIFO (0..3 = 8,16,32,56 spaces)
  3    Reserved
  2    TX FIFO Reset (0=No change, 1=Clear TX FIFO)
```

```
1    RX FIFO Reset (0=No change, 1=Clear RX FIFO)
0    RX/TX FIFO enable (0=Disable, 1=Enable)
```

Bit1,2: FIFO reset requires at least two XTAL1 clocks, therefore, they cannot be reset without the presence of the XTAL1 clock.
Bit4-5: Can only be modified and enabled when EFR[4] is set. This is because the transmit trigger level is regarded as an enhanced function.

IR[18h] - LCR - Line Control Register (R/W)

```
7    Divisor latch access enable (DLAB) (0=Normal, 1=Access DLL/DLH)
6    Break control bit   (0=Off/Normal, 1=Break/Force TX to logic 0)
5-4  Parity Type         (0=Odd, 1=Even, 2=Forced1, 3=Forced0)
3    Parity Enable       (0=Disable, 1=Enable)
2    Number of stop bits (0=1bit, 1=2bit, or 1.5bit in 5bit mode)
1-0  Word length         (0=5bit, 1=6bit, 2=7bit, 3=8bit)
```

IR[28h.R] - LSR - Line Status Register (R)

```
7    FIFO data error (0=None, 1=At least one Error/Break anywhere in RX FIFO)
6    THR and TSR empty   (0=Not empty, 1=Empty)  (TX FIFO and TX shift reg)
5    THR empty           (0=Not empty, 1=Empty)  (TX FIFO)
4    Break interrupt     (0=No, 1=Break, next RX FIFO char reads as 00h)
3    Framing Error       (0=No, 1=Error; bad stop bit in data from RX FIFO)
2    Parity Error        (0=No, 1=Error; in data being read from RX FIFO)
1    Overrun Error       (0=No, 1=Error; overrun has occurred)
0    Data in Receiver    (0=No, 1=At least one character in the RX FIFO)
```

Bit2-4 reflect the error bits (BI, FE, PE) of the character at the top of the RX FIFO (next character to be read). Therefore, errors in a character are identified by reading the LSR and then reading the RHR.

IR[20h] - MCR - Modem Control Register (R/W)

```
7    Clock Divisor    (0=Div1 clock input, 1=Div4 clock input)
6    IrDA mode enable (0=Normal UART mode, 1=IrDA mode) (aka pulse RX/TX?)
5    Xon Any function (0=Disable, 1=Enable)
4    Enable Loopback  (0=Normal, 1=local Loopback mode, internal)
      In this mode the MCR[1:0] signals are looped back into MSR[4:5] and
      the TX output is looped back to the RX input internally.
3    Reserved
2    TCR/TLR register access enable (0=Disable, 1=Enable)
1    RTS output (if Auto RTS disabled)        (0=High/Inactive, 1=Low/Active)
      In Loopback mode, controls MSR[4].
0    DTR output (GPIO5) (if IOControl.bit1=1) (0=High/Inactive, 1=Low/Active)
      Writing to IOState bit 5 will not have any effect on this pin.
```

Bit2,5-7 can only be modified when EFR[4] is set, ie. EFR[4] is a write enable.
Bit0: Only available on SC16IS750/SC16IS760.

IR[30h.normal.R] - MSR - Modem Status Register (R)

```
7    CD input state (inverted, 1=Not HIGH, or so)     ;\only if modem mode
6    RI input state (inverted, 1=Not HIGH, or so)     ; (IOControl.bit1=1)
5    DSR input state (inverted, 1=Not HIGH, or so)    ;/
4    CTS input state (inverted, 1=Not HIGH, or so)
3    CD input changed state                  (0=No, 1=Yes)    ;\
2    RI input changed state from LOW to HIGH (0=No, 1=Yes)    ; cleared
1    DSR input changed state                 (0=No, 1=Yes)    ; on read
0    CTS input changed state                 (0=No, 1=Yes)    ;/
```

Bit1-3,5-7: Only available on SC16IS750/SC16IS760.
Remark: The primary inputs RI, CD, CTS, DSR are all active LOW.
Reading IOState bit 6,6,4 does not reflect the true state of CD,RI,DSR pin, uh?

IR[08h] - IER - Interrupt Enable Register (R/W)

```
7    CTS change interrupt enable         (0=Disable, 1=Enable)
6    RTS change interrupt enable         (0=Disable, 1=Enable)
5    Xoff received interrupt             (0=Disable, 1=Enable)
4    Sleep mode                          (0=Disable, 1=Sleep Mode)
```

```
  3    Modem Status Interrupt               (0=Disable, 1=Enable)
  2    Receive Line Status error interrupt  (0=Disable, 1=Enable)
  1    Transmit Holding Register interrupt  (0=Disable, 1=Enable)
  0    Receive Holding Register interrupt   (0=Disable, 1=Enable)
```
Bit4-7 can only be modified if EFR[4] is set, that is, EFR[4] is a write enable. Re-enabling IER[1] will not cause a new interrupt if the THR is below the threshold.
Bit3: Only available on the SC16IS750/SC16IS760.
Bit3: See IOControl.bit1 for the description of how to program the pins as modem pins.

IR[10h.R] - IIR - Interrupt Identification Register (R)
```
  7    Mirror of FCR.bit0: RX/TX FIFO enable (0=Disable, 1=Enable) ;\both same
  6    Mirror of FCR.bit0: RX/TX FIFO enable (0=Disable, 1=Enable) ;/
  5-1  Interrupt source (5bit encoded, see below) (valid when bit0=0)
  0    Interrupt status (0=IRQ pending, 1=No IRQ pending)
```
Interrupt sources:
```
  Src Prio Expl.
  03h 1    Receiver Line Status error
  06h 2    Receiver time-out interrupt
  02h 2    RHR interrupt
  01h 3    THR interrupt
  00h 4    MSR Modem interrupt           (SC16IS750/SC16IS760 only)
  18h 5    IOState GPIO input pin change (SC16IS750/SC16IS760 only)
  08h 6    received Xoff signal/special character
  10h 7    CTS, RTS change state from active(LOW) to inactive(HIGH)
```
Burst reads on the serial interface should not be performed on the IIR register (ie. don't read multiple elements on the I2C-bus without a STOP or repeated START condition, and don't read multiple elements on the SPI bus without de-asserting the CS pin).

IR[10h.LCR=BFh] - EFR - Enhanced Features Register (R/W)
```
  7    CTS flow control enable (0=Disable, 1=Enable; stop TX upon CTS=HIGH)
  6    RTS flow control enable (0=Disable, 1=Enable; change RTS upon TCR)
         RTS pin goes HIGH when RX FIFO halt trigger level TCR[3:0] is reached
         RTS pin goes LOW when RX FIFO resume trigger level TCR[7:4] is reached
  5    Special character detect (0=Disable, 1=Enable, compare with Xoff2)
         If a Xoff2 match occurs, the received data is transferred to FIFO and
         IIR.bit4 is set to 1 to indicate a special character has been detected
  4    Enhanced functions enable bit (0=Disable, 1=Enable)
         Enables writing to IER[7:4], FCR[5:4], MCR[7:5]
  3-0  Combinations of software flow control can be selected by programming
         these bits. See Table 3 "Software flow control options (EFR[3:0])".
```

IR[00h.DLAB=1] - DLL - Baudrate Divisor Latch Register Low, bit0-7 (R/W)
IR[08h.DLAB=1] - DLH - Baudrate Divisor Latch Register High, bit8-15 (R/W)
```
  15-0  16bit divisor for generation of the clock in the baudrate generator
```
Remark: DLL and DLH can only be written to before Sleep mode is enabled, that is, before IER[4] is set.

IR[30h.MCR[2]=1 and EFR[4]=1] - TCR - Transmission Control Register (R/W)
```
  7-4  RX FIFO flow control resume trigger level  (N*4) (0..14 = 0..56 chars)
  3-0  RX FIFO flow control halt trigger level    (N*4) (1..15 = 4..60 chars)
```
RX FIFO threshold levels to stop/start transmission during hardware/software flow control (ie. for RTS pin, or Xon/Xoff chars).
Remark: TCR can only be written to when EFR[4]=1 and MCR[2]=1.
The programmer must program the TCR such that Halt>Resume, aka TCR[3:0]>TCR[7:4]. There is no built-in hardware check to make sure this condition is met.
Also, the TCR must be programmed with this condition before auto RTS or software flow control is enabled to avoid spurious operation of the device.

IR[38h.MCR[2]=1 and EFR[4]=1] - TLR - Trigger Level Register (R/W)
```
  7-4  RX FIFO interrupt trigger level (N*4) (1..15 = 4..60 chars available)
```

```
  3-0  TX FIFO interrupt trigger level (N*4) (1..15 = 4..60 spaces available)
```
Remark: TLR can only be written to when EFR[4]=1 and MCR[2]=1. If TLR[3:0] or TLR[7:4] are logical 0, the selectable trigger levels via the FIFO Control Register (FCR) are used for the transmit and receive FIFO trigger levels. Trigger levels from 4 characters to 60 characters are available with a granularity of four. The TLR should be programmed for NÏ4, where N is the desired trigger level.
When the trigger level setting in TLR is zero, the SC16IS740/750/760 uses the trigger level setting defined in FCR. If TLR has non-zero trigger level value, the trigger level defined in FCR is discarded. This applies to both transmit FIFO and receive FIFO trigger level setting.
When TLR is used for RX trigger level control, FCR[7:6] should be left at the default state, that is, '00'.

IR[40h.R] - TXLVL - Transmitter FIFO Level register (R)
```
  7    Unused (0)
  6-0  Number of spaces available in TX FIFO (00h..40h; 00h=Full, 40h=Empty)
```

IR[48h.R] - RXLVL - Receiver FIFO Level register (R)
```
  7    Unused (0)
  6-0  Number of characters stored in RX FIFO (00h..40h; 00h=Empty, 40h=Full)
```

IR[50h] - IODir - Programmable I/O pins Direction register (R/W)
This register is only available on the SC16IS750 and SC16IS760.
```
  7-0  IODir set GPIO pins [7:0] to input or output (0=Input, 1=Output)
```
Remark: If there is a pending input (GPIO) interrupt and IODir is written, this pending interrupt will be cleared, that is, the interrupt signal will be negated.

IR[58h] - IOState - Programmable I/O pins State Register (R/W)
```
  7-0  IOState Write this register: set the logic level on the output pins
         0=set output pin to zero
         1=set output pin to one
       Read this register: return states of all pins
```
This register is only available on the SC16IS750 and SC16IS760.
When read, this register returns the actual state of all I/O pins.
When write, each register bit will be transferred to the corresponding IO pin programmed as output.

IR[60h] - IOIntEna - I/O Interrupt Enable Register (R/W)
```
  7-0  Input change interrupt enable (0=Disable, 1=Enable)
```
This register is only available on the SC16IS750 and SC16IS760.
This register enables the interrupt due to a change in the I/O configured as inputs. If GPIO[7:4] are programmed as modem pins, their interrupt generation must be enabled via IER register bit3. In this case bit4-7 of IOIntEna will have no effect on GPIO[7:4].

IR[70h] - IOControl - I/O Control register (R/W)
```
  7-4  Reserved
  3    Software reset (0=No change, 1=Reset)                    (W)
  2    Reserved
  1    Use GPIO[7:4] as GPIO or as modem pins (0=GPIO, 1=RI,CD,DTR,DSR)
  0    IOLATCH enable/disable inputs latching
       0 = input values are not latched. A change in any input generates an
         interrupt. A read of the input register clears the interrupt. If the
         input goes back to its initial logic state before the input register
         is read, then the interrupt is cleared.
       1 = input values are latched. A change in the input generates an
         interrupt and the input logic value is loaded in the bit of the
         corresponding input state register (IOState). A read of the IOState
         register clears the interrupt. If the input pin goes back to its
         initial logic state before the interrupt register is read, then the
         interrupt is not cleared and the corresponding bit of the IOState
         register keeps the logic value that initiates the interrupt.
```

This register is only available on the SC16IS750 and SC16IS760.
Remark: As I/O pins, the direction, state, and interrupt of GPIO4 to GPIO7 are controlled by the following registers: IODir, IOState, IOIntEna, and IOControl. The state of CD, RI, DSR pins will not be reflected in MSR[7:5] or MSR[3:1], and any change of state on these three pins will not trigger a modem status interrupt (even if enabled via IER[3]), and the state of the DTR pin cannot be controlled by MCR[0].
As modem CD, RI, DSR pins, the status at the input of these three pins can be read from MSR[7:5] and MSR[3:1], and the state of DTR pin can be controlled by MCR[0]. Also, if modem status interrupt bit is enabled, IER[3], a change of state of RI, CD, DSR pins will trigger a modem interrupt. Bit[7:4] of the IODir, IOState, and IOIntEna registers will not have any effect on these three pins.

IR[78h] - EFCR - Extra Features Control Register (R/W)

```
  7    IrDA pulse mode (0=Max115kbit/s, ratio 3/16, 1=Max1152kbit/s, ratio 1/4)
  6    Reserved
  5    Invert RTS signal in RS-485 mode (0=RTS=0 during TX, 1=RTS=1 during TX)
  4    Enable the transmitter to control the RTS pin (0=Disable, 1=Enable)
  3    Reserved
  2    Disable transmitter  (0=Normal, 1=Disable; stop forwarding THR to TSR)
        UART does not send serial data out on the transmit pin, but the
        transmit FIFO will continue to receive data from host until full.
        Any data in the TSR will be sent out before the transmitter goes into
        disable state.
  1    Disable receiver     (0=Normal, 1=Disable)
        UART will stop receiving data immediately once this bit set to 1, and
        any data in the TSR?? will be sent to the receive FIFO.
        User is advised not to set this bit during receiving.
  0    Enable 9bit or Multidrop mode  (0=Normal RS-232, 1=Enable RS-485)
```

Bit7 is SC16IS760 only.

_____ Infrared IrDA Notes _____

Infrared Usage in 3DS

The 3DS doesn't have any expansion connector, and it's instead using the IrDA port for expansion hardware (with external batteries):

- Circle Pad Pro (a second Circle Pad, attached right of the A/B/X/Y buttons)
- NFC Near-field communication (for Amiibo figures)

For the NFC thing, see here:
3DS NFC Adapter
That add-on hardware exists for Old3DS only. The New3DS does have equivalent hardware built-in, and thus doesn't need that add-ons.
Additionally, IrDA could be used for things like file transfers, unknown if there are 3DS tools supporting such things?

Infrared Signals

The SC16IS750 UART/IrDA SIR datasheet does mainly describe the UART part, and barely mentions infrared at all (apart from saying that the chip is IrDA SIR compatible, and that IrDA can be enabled in MCR.bit6).
Going by other sources, IrDA SIR works like "UART over infrared". In lack of further details, here some guesses on how it might work (and how far it is (or isn't) compatible with regular cable-based UARTs):

- RX/TX are probably using pulsed signals instead of constant LOW/HIGH levels?
- RTS/CTS/DTR/DSR are probably not implemented at all?
- Simultaneous RX+TX might work in opposite light-direction, if no reflection?
- Infrared interruptions and noise may require additional error checking?
- Infrared at less than 10mm (as in Circle Pad Pro) may work 100% error-free?

Infrared Protocol

Specifications for the IrDA standard aren't available for free, and there's little known about the protocol:

- IrDA protocol supports file transfers from one device to another
- IrDA protocol must include some sort of packet headers and checksums
- And probably much stuff

However, it's unknown if Nintendo did bother to follow the IrDA standard in their Circle Pad Pro and NFC add-ons. It's also possible that they have just used a custom UART-like transfer protocol for that hardware.

Infrared Range
IrDA is supposed for short-range transfers (about 20cm - 100cm) in binary format. As such, it isn't intended to use the hardware for things like classic TV remote controls (although... maybe it could be tweaked to output TV remote signals on TX pin... possibly in UART mode with IrDA disabled?).

3DS I2C LCD Screen Controllers

I2C Bus/Device 1:2Ch - Upper LCD screen (lcd0)
I2C Bus/Device 1:2Eh - Lower LCD screen (lcd1)
LCD controllers for top/bottom screen.

I2C Access
I2C writing works as for most other I2C chips:
```
I2cWriteCmd (Device+0, Index, Data)
```
However, I2C reading works uncommon:
```
I2cWriteCmd (Device+0, 40h, ReadIndex)
I2cReadCmd  (Device+1, ReadIndex, Data)
```
Ie. the LCD chips distinguish between WriteIndex and ReadIndex, with the ReadIndex being set by writing to register 40h. The ReadIndex value in the response can be ignored (or verified to be same as the written ReadIndex value).

I2C Registers
```
00h      Zero                                                        (-)
01h      Display on/off (10h=Display on, 11h=Display black) ;bit0,4 is R/W
02h      Usually 01h, but bit0 is R/W
03h      Usually 00h, but bit0-7 is R/W
04h..10h Zero                                                        (-)
11h      Whatever (set to 10h during init, maybe bit4=reset?)
12h..3Fh Zero                                                        (-)
40h      Read Index for I2C read (00h..FFh)
41h..4Fh Zero                                                        (-)
50h      Bit0 is R/W  (set to 1 for top-screen; left 0 on bottom-screen)
51h..53h Zero                                                        (-)
54h      Usually 00h, but bit0 is R/W
55h      Usually 00h, but bit0-1 is R/W
56h..5Fh Zero                                                        (-)
60h      Whatever (set to 00h during init)  ;bit0 is R/W ?
61h      Status (initially B4h=Top Screen, F8h=Bottom Screen)        (R?)
62h      Status (01h=Ready)  (except, broken top screen reads 00h?)  (R?)
63h      Zero                                                        (-)
64h      Whatever/Fixed 03h                                          (R)
65h..AEh Zero                                                        (-)
AFh      Usually 00h, but bit0-7 is R/W
B0h..D3h Zero                                                        (-)
D4h      Usually 00h, but lower 4bit are R/W
D5h      Usually 00h, but bit0-7 is R/W
D6h..FDh Zero                                                        (-)
FEh      Usually 00h, but bit0-7 is R/W
FFh      Maybe Chip ID (10h=Top Screen, C7h=Bottom Screen)           (R)
```
Odd effect observed during testing: Setting LCD[AFh]=AAh causes MANY more registers to become nonzero; this doesn't happen for other values like LCD[AFh]=0Fh,F0h,FFh.

bootrom error screen init sequence for the I2C registers:
```
LCD[11h]=10h     ;whatever
LCD[50h]=01h     ;whatever, this one done for TOP screen only
```

```
  LCD[60h]=00h     ;whatever
  LCD[01h]=10h/11h ;on/off (10h=display on, 11h=display black)
```
The bootrom additionally checks if status LCD[62h]=01h=Ready.

3DS I2C New3DS Near-Field Communication (NFC)

I2C Bus/Device 1:EEh - Near-Field Communication (NFC) (New3DS-only)
This device is transferring command/response packets on the I2C bus (so one must only transfer Device+Data bytes; without any register Index values).

Datasheets and Specs
The NFC controller appears to be a 32pin Broadcom BCM20791 chip, there is no datasheet for that chip, however, a datasheet for 34pin BCM20793S chip does exist, but unfortunately without any specs on the transfer protocol.
Specifications for the lower-level transfer protocol (and maybe even the I2C bus protocol?) are made by the "NFC Forum" (a non-profit organization that charges lots of money for a copy of those specs).
There is some android open source code for "bcm2079x" available.

New3DS Transfer Protocol...
Command request / response structure:
```
  00h 1     Normally 10h?
  01h 1     Command source / destination
  02h 1     CmdID
  03h 1     Payload size (LEN)
  04h LEN   Data/parameters
```

The command response payload is usually at least 1-byte, where that byte appears to be normally 00h. For command requests the payload data is the command parameters.

For command requests sent to the NFC tag itself, Cmd[1]=0x0 and CmdID=0x0. The command request payload data here is the actual command request data for the NFC tag, starting with the CmdID u8 at payload+0.

During NFC module startup, a certain command (?) is sent to the controller which eventually (after various cmd-reply headers etc) returns the following after the first byte in the payload:
```
  000000: 44 65 63 20 32 32 20 32 30 31 32 31 34 3a 35 33  Dec 22 201214:53
  000010: 3a 35 30 01 05 0d 46 05 1b 79 20 07 32 30 37 39  :50...F..y .2079
  000020: 31 42 35                                         1B5
```
Or that is: "Dec 22 201214:53:50<binary>20791B5". Therefore, this appears to return the part-number of the NFC controller (other command request(s) / response(s) use this part-number value too).
Actually, 20791B5 seems to be NFC certification number for an unspecified Broadcom product (or possibly 20791B5 is the product name itself... yes, apparently it's Broadcom BCM20791).
Protocol = NFC Forum NFC Controller Interface (NCI) for host interface

NFC controller commands
```
  CmdRequest[1, uh?]  CmdID   Payload data for parameters
  2Eh                 2Fh     Firmware image for this chunk, size varies.
```
This is used during NFC module startup to upload the firmware image to the NFC controller. This is used repeatedly to upload multiple chunks of the image.

Power-up
After power-up, the NFC chip automatically sends these bytes upon I2C reads:
```
  10 60 00 02 00 01     ;followed by endless FF bytes
```
That seems to be a 4-byte header, with 2-byte data, as indicated in hdr[3]=02h.

NFC Interrupt

Triggers when there is data available for reading. Unknown if there is also a way for determining if the thing is ready for writing a new command... maybe one should simply wait for a response packet after sending any command packets?

Old3DS Version
The Old3DS did have an external NFC Adapter add-on, accessed via IrDA.
3DS NFC Adapter
The 3DS OS is supposedly having backwards/forwards compatible NFC functions, but the underlying transfer protocol seems to be completely different (relying on IrDA to communicate with the adpater firmare, instead of directly accessing the BCM20791 chip).

3DS NFC Adapter

The Nintendo NFC adapter, formally Nintendo NFC Reader/Writer and codenamed Fangate, is an external device which adds NFC capabilities for amiibos to old Nintendo 3DS and Nintendo 2DS consoles, using the infrared port on the back of the console.

It launched simultaneously with Animal Crossing Happy Home Designer, with which it's optionally bundled; it can also be bought standalone at a nominal(?) price of 21 EUR.

Technical details
Based on analysis of the fangate_updater.bin file, which is part of the old Nintendo 3DS operating system since 9.3.0-21 and contains the firmware running on the external adapter; and analysis of the NFC Services running on old 3DS.

```
  SOC inside the adapter: Broadcom BCM20791B1
                          or ST proprietary "MCU-FGT/rev.A/GH24S VQ"
                            uh, either or? or rather both?
                            is that guessed from fangate_updater.bin,
                            or seen on photos of the actual pcb/hardware?
  CPU:            ARM Cortex M0
```

Memory map:

```
  Address    Size    Description
  08008000h  256KB?  Firmware (fangate_updater.bin)
  20000000h  128KB?  RAM
  40023C00h  1Ch     FLASH ROM control
  E000ED00h  104h    ARM Cortex system control block
```

Layer 1 - IR communications framing format
Packets are sent using IrDA-SIR (using ir:USER) at 115200 bps 8N1 (eight data bits, no parity, one stop bit).

```
 IR framing format - short frame (max 3Fh data bytes):
  00h   1  Synchronization byte (A5h)
  01h   1  Reserved for future use (00h)
  02h   1  bit7:RFU (0), bit6:Short frame (0), bit0-5:Payload size
  03h   N  Payload
  03h+N 1  CRC-8-CCITT for whole packet [00h..N+02h]  ;uh, before/after XORing?
 IR framing format - long frame (max 3FFFh data bytes):
  00h   1  Synchronization byte (A5h)
  01h   1  Reserved for future use (???)
  02h   1  bit7:RFU (0), bit6:Long frame (1), bit0-5: Payload size upper bits
  03h   1  Payload size (lower 8 bits)
  04h   N  Payload (XOR-encrypted)
  04h+N 1  CRC-8-CCITT for whole packet [00h..N+03h]  ;uh, before/after XORing?
```

The payload is encrypted using a XOR-based encryption:

```
    halfCount = size/2          ;Divide by 2 rounding towards zero
    xorval = htobe16(0xE963)    ;that is, BIG-ENDIAN ?
    for (i = 0; i < halfCount; i++)
```

```
xorval = xorval XOR = src[i]
dst[i] = xorval
```

Layer 2 - "ircom"

ircom is a simple stateful point-to-point master-slave communication protocol built on top of IR layer 1.

```
00h 4   Random (makes the whole packet look random after XOR encryption)
04h 1   bit4-7:RFU?, bit0-3:Protocol version (01h)
05h 1   Connection ID of master (3DS), value determined by master
06h 1   Connection ID of slave (NFC adapter), value determined by slave
07h 1   bit4-7:???, bit0-3: Operation code
08h N-8 Payload (if any)
```

Operation codes:

Code	Name	Has payload	Direction
00h =	Layer 3 command	Yes	Master to slave
01h =	???		
02h =	???		
03h =	???		
04h =	???		
05h =	???		
06h =	???		
07h =	???		
08h =	???		
09h =	???		
0Ah =	Disconnect request	No	Master to slave
0Bh =	Disconnection acknowledgment	No	Slave to master
0Ch =	Handshake	No	Master to slave
0Dh =	Handshake acknowledgement	No	Slave to master
0Eh =	???		
0Fh =	???		

NFC adapter will ignore packets whose protocol version is not 1. It will not even reply.
Connection identifier is a random byte the 3DS assigns to identify the connection should there be several connections in range at once. Slave devices must save this value from the initial handshake packet and use it for replies. The 3DS will also save the connection identifier byte of the slave which is also random. The 3DS must also ignore packets whose connection ID does not match.

Layer 3

Layer 3 is the payload of layer 2. A lot is unknown of this layer and thus a lot of assumptions were made in the following tables.

Layer 3 contains the following data:

```
00h    bit4-7:Request identifier nibble, bit0-3: Always 01h
01h    Slave/master identifier byte
02h    Request type code upper byte
03h    Request type code lower byte
04h+   Payload (if any)
```

Request identifier nibble is incremented by 0x1 at every new request by the master, the same value for this byte is also sent by the slave in response to the request of the master
Slave/master identifier byte is 0x1 for a message from the master and 0x10 for a message from the slave

Request type codes

Code	Request description	Direction	Has payload
0000h =	ACK	Slave to master	Yes
0003h =	Get firmware version and battery level	Master to slave	No
0004h =	Firmware version and battery level	Slave to master	Yes
0100h =	Unknown, slave always responds with ACK	Master to slave	Yes
0202h =	Request to stop communication with NFC tag	Master to slave	No
0204h =	Get dumped data from NFC tag	Master to slave	No
0205h =	Data from NFC tag	Slave to master	Yes
0206h =	Request to start communication with NFC tag	Master to slave	Yes
0207h =	Request to write data to NFC tag	Master to slave	Yes

0000h
Acknowledgement message always send by slave. Payload always contains 000000AAh.

0004h
Payload contains firmware version and battery level of NFC adapter. Payload has a size of 6 bytes.

```
00h    Upper or lower byte of version number, newest firmware is 01h
01h    Upper or lower byte of version number, newest firmware is 06h
02h    Padding byte? Always 00h
03h    Padding byte? Always 00h
04h    Battery level (03h=Full, 00h=Empty)
05h    Padding byte? Always 00h
```

NFC reader LED already turns red when the battery level byte is 02h, this will also trigger a low battery level warning on the 3DS.

0100h
The purpose of this request by the master is unknown. Slave always responds with ACK. Payload of this request is always 0003E8AAh.

0205h
Payload contains data regarding NFC communication. The first byte of the payload means the following:

```
Code   Description
00h =  NFC communication is stopped as result of a 0202h request from master
01h =  No NFC tag on top of the reader
02h =  Busy dumping NFC tag
03h =  NFC tag dump after write by master
04h =  NFC tag fully dumped
05h =  NFC tag dump after write by master
07h =  NFC tag not a NTAG215 or contains no Amiibo compatible data
08h =  NFC tag removed from reader
```

After the tag is written by the master and dumped again, the first few dumps start with 03h, this changes to 05h after a few dumps. The reason for this is unknown.

0206h
Request from master to start NFC communication. Payload always contains 19 00h padding bytes, followed by C80300393A737486000001h and another 26 00h padding bytes.

0207h
Request from master to write to NFC tag. The request packet already contains the desired data to be written to the tag. Payload start with two 00h padding bytes followed by the 7 ID bytes of the tag. These ID bytes are used by the NFC adapter to check if same Amiibo is placed on the NFC adapter again.

Samples
NFC handshake beacons:

```
Layer 1 packet                          Layer 2 packet       Layer 3 packet
A5 00 08 73 FE A5 C4 A4 2C A4 20 F5     9A 9D D6 3A 01 E8 00 0C         ?
A5 00 08 D1 3E B7 7B B6 91 B6 9D 87     38 5D 66 45 01 EA 00 0C         ?
A5 00 08 09 58 23 36 22 DA 22 D6 AE     E0 3B 2A 6E 01 EC 00 0C         ?
A5 00 08 5E DD A4 A0 A5 4E A5 42 A8     B7 BE FA 7D 01 EE 00 0C         ?
A5 00 08 BC 19 C6 37 C7 C7 C7 CB 8B     55 7A 7A 2E 01 F0 00 0C         ?
A5 00 08 C9 15 F6 63 F7 91 F7 9D B2     20 76 3F 76 01 F2 00 0C         ?
A5 00 08 6E 48 47 1A 46 EE 46 E2 C7     87 2B 29 52 01 F4 00 0C         ?
A5 00 08 A2 8C E5 C3 E4 35 E4 39 74     4B EF 47 4F 01 F6 00 0C         ?
A5 00 08 26 1C 07 10 06 E8 06 E4 64     CF 7F 21 0C 01 F8 00 0C         ?
A5 00 08 7E 73 A2 3F A3 C5 A3 C9 FD     97 10 DC 4C 01 FA 00 0C         ?
A5 00 08 75 00 F3 B8 F2 44 F2 48 63     9C 63 86 B8 01 FC 00 0C         ?
A5 00 08 8D AC 0F D5 0E 2B 0E 27 72     64 CF 82 79 01 FE 00 0C         ?
A5 00 08 A3 55 7C 53 7D 52 7D 5E B2     4A 36 DF 06 01 01 00 0C         ?
A5 00 08 15 06 43 C0 42 C3 42 CF 85     FC 65 56 C6 01 03 00 0C         ?
```

```
A5 00 08 66 E0 9A 17 9B 12 9B 1E A0     8F 83 FC F7 01 05 00 0C         ?
A5 00 08 A4 35 09 97 08 90 08 9C 25     4D 56 AD A2 01 07 00 0C         ?
A5 00 08 73 E2 BD AF BC A6 BC AA 60     9A 81 CE 4D 01 09 00 0C         ?
A5 00 08 02 57 D7 B0 D6 BB D6 B7 28     EB 34 D5 E7 01 0B 00 0C         ?
A5 00 08 0D 79 01 AA 00 A7 00 AB 22     E4 1A 0C D3 01 0D 00 0C         ?
A5 00 08 14 91 04 B9 05 B6 05 BA B2     FD F2 10 28 01 0F 00 0C         ?
A5 00 08 2C 86 B1 49 B0 58 B0 54 C0     C5 E5 9D CF 01 11 00 0C         ?
A5 00 08 D5 1D DE DB DF C8 DF C4 F9     3C 7E 0B C6 01 13 00 0C         ?
A5 00 08 AF 75 DE 5C DF 49 DF 45 9C     46 16 71 29 01 15 00 0C         ?
A5 00 08 C8 E2 5B C6 5A D1 5A DD B5     21 81 93 24 01 17 00 0C         ?
A5 00 08 9B 51 68 2D 69 34 69 38 41     72 32 F3 7C 01 19 00 0C         ?
A5 00 08 13 7B 9F EF 9E F4 9E F8 32     FA 18 8C 94 01 1B 00 0C         ?
A5 00 08 A7 62 02 9C 03 81 03 8D BD     4E 01 A5 FE 01 1D 00 0C         ?
A5 00 08 39 06 94 36 95 29 95 25 09     D0 65 AD 30 01 1F 00 0C         ?
A5 00 08 32 4C D7 C0 D6 E1 D6 ED 92     DB 2F E5 8C 01 21 00 0C         ?
A5 00 08 83 BE F2 8F F3 AC F3 A0 B1     6A DD 71 31 01 23 00 0C         ?
A5 00 08 83 5E A0 57 A1 72 A1 7E F0     6A 3D 23 09 01 25 00 0C         ?
A5 00 08 6E C8 AD 69 AC 4E AC 42 D1     87 AB C3 A1 01 27 00 0C         ?
A5 00 08 C7 33 A1 2C A0 05 A0 09 FC     2E 50 66 1F 01 29 00 0C         ?
```

External links
BCM2079x brief on Broadcom's website
Python scripts to sniff and spoof IR communication between the NFC reader and 3DS using an IrDA adapter

3DS I2C New3DS C-Stick and ZL/ZR-Buttons

I2C Bus/Device 2:54h - C-Stick and ZL/ZR-Buttons
This is a New3DS-only device (the Old3DS does have a similar add-on: Circle Pad Pro, connected to the infrared expansion port).
Oberseve that the bootrom doesn't initialized I2C_BUS2_CNTEX/SCL (especially CNTEX is required to have "Wait if SCL held low" enabled, else the C-stick replies are garbage).
The chip triggers ARM11 IRQ 68h (aka GPIO_DATA3.bit0=0) when moving the analog input or buttons.
Reading the I2C data changes GPIO back to bit0=1.
Maybe this is the HUGE 44pin HF374 chip on button board, but if it's merely used for ZL/ZR and C-stick, why is that chip so huge?

I2C Read(Device+1,Byte0,Byte1,Byte2,Byte3,...)
Reading returns some kind of array, always starting with the status value in byte0, there is no need to write an index value before reading.
The array has useful info stored in first 4-7 bytes; and internal/garbage when reading further bytes, up to including something that looks like a CPU stack).

```
 The array entries are...
  00h       Status byte (80h..83h, or FFh) (power-up default=80h)
  01h       Button byte (00h=None, bit2=ZL, ?=ZR)
  02h       Analog X (00h=Center, -1xh=Left, +1xh=Right) ;\if enabled
  03h       Analog Y (00h=Center, -1xh=Down, +1xh=Up)    ;/
 Following bytes aren't needed, except for better 8bit X/Y resolution...
  04h       Fixed FFh
  05h       Analog X (00h=Center, -7xh=Left, +7xh=Right) ;\hires
  06h       Analog Y (00h=Center, -7xh=Down, +7xh=Up)    ;/
 Following bytes aren't actually useful...
  07h       Fixed 00h
  08h       Analog X (7xh=Center, FEh=Left, 00h=Right)   ;\unsigned/uncentered
  09h       Analog Y (7xh=Center, FEh=Down, 00h=Up)      ;/
  0Ah       Fixed 00h
  0Bh       Analog X (7xh=Center, FEh=Left, 00h=Right)   ;\same as [08h,09h]
  0Ch       Analog Y (7xh=Center, FEh=Down, 00h=Up)      ;/
  0Dh       Fixed 00h
  0Eh       Center X (7xh)  ;\auto-calibrating, with minor changes
```

```
 0Fh       Center Y (7xh)  ;/every some seconds
 10h       Fixed 00h
 11h..15h  Analog stuff
 16h       Flag (00h=Idle, 80h=Analog is/was recently touched)
 17h       Analog stuff
 18h       Flag (01h=Idle, 02h=Analog is/was recently left or down)
 19h       Analog stuff
Following bytes can crash the chip upon reading (see below for details):
 1Ah       Flag (01h=Idle, 00h/02h=Analog is/was recently somehow moved)
 1Bh       Fixed 01h
 1Ch       Fixed 11h
 1Dh       Historic X (7xh)  ;\updated every some seconds
 1Eh       Historic Y (7xh)  ;/(same as center when idle)
 1Fh..27h  Fixed 00h-filled
 28h..2Ch  Fixed 07h,06h,06h,03h,01h
 2Dh..3Bh  Fixed 00h-filled
 3Ch       Fixed 01h
 3Dh..46h  Fixed 00h-filled
 47h..4Bh  Fixed 04h,C0h,00h,00h,03h
 4Ch       Whatever, changes
 4Dh       Flag (00h=Idle, 01h=Analog is/was recently moved)
 4Eh..54h  Fixed 01h,00h,06h,01h,A5h,00h,00h
 55h       Initially random xxh, becomes 00h/01h after button/analog
 56h       Flag 00h/10h/20h
 57h       Analog X (00h=Center, -1xh=Left, +1xh=Right) ;\same as [02h,03h]
 58h       Analog Y (00h=Center, -1xh=Down, +1xh=Up)    ;/
 59h       Analog
 5Ah       Analog
 5Bh..65h  Fixed 07h,00h,08h,04h,00h,00h,00h,09h,22h,71h,00h
 66h       Fast Timer (00h..1xh or so)
 67h       Fixed 15h ;maybe update period in ms, maybe limit for above timer?
 68h       Slow Timer (00h..08h, increasing)
 69h..6Ch  Fixed 09h,03h,00h,00h         ;maybe 09h is limit for above timer?
 6Dh       Up/down Timer (00h=Idle, increases-then-decreases upon analog move)
 6Eh..6Fh  Fixed 00h,00h
 70h       Button byte (00h=None, bit4=ZL, ?=ZR)
 71h       Usually 00h (sometimes shortly 10h or so)
 72h..74h  Fixed 00h-filled
 75h       Fixed FFh
 76h..7Bh  Fixed 00h-filled
 7Ch..82h  Fixed 7Fh,15h,09h,03h,54h,28h,10h
 83h..BDh  Fixed 00h-filled
Following bytes might be CPU stack, first some bytes maybe random/garbage?
 BEh       7Ah,02h,1Dh,D9h,C7h,93h,31h,CCh,7Eh,A9h,BEh,86h,B3h,93h,6Dh,07h
 CEh       C7h,82h,E7h,00h,10h,00h,00h,10h,00h,10h,00h,10h,xxh,xxh,xxh,xxh
 DEh       xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
Following bytes seem to be used...
 EEh..xxxx Fixed FFh-filled (somewhat endless repeating)
```

Caution: The chip hangs after some seconds when reading more than 1Ah bytes; at that point all values get frozen (except, the "stack" at DDh..EDh does keep changing).

I2C Write(Device,Mode)

Writing does set a mode value, eg. write(Device,Mode). There is no need to write anything, the analog input and irq are automatically enabled on power-up. Effects for different mode values are:

```
 None      Default is Status=80h at power-up
 00h..51h  Set Status=80h
 52h..F4h  Set Status=81h
 F5h       Set Status=82h
 F6h..F9h  Set Status=83h
 FAh..FBh  Set Status=81h
 FCh..FDh  Set Status=FFh
 FEh       Set Status=80h with long 1 second I2C-clk-hold delay?
 FFh       Set Status=80h
```

Unknown what those mode/status values are meaning exactly. Maybe some can change polling interval or enable/disable interrupts, or maybe even do dangerous stuff like reflashing the firmware?
Some of the mode values disable the Analog bytes at index 02h/03h (causing that bytes to become always 00h, or maybe actually divide them by a large value, causing them to be always NEAR 00h?)
Trying to write more bytes (eg. write(Device,xx,yy) seems to apply the last written byte as mode value.

Operating System
The 3DS OS does access the C-stick via IR:RST service (probably for backwards compatibility with Circle Pad Pro, which was accessed via infrared).
The OS can reportedly change the update period in range of 10..21ms (aka 0Ah..15h), maybe via whatever I2C write.
The OS does reportedly return analog values in range -9Ch..+9Ch, maybe that are the I2C values scaled for Circle Pad Pro compatibility, or maybe a sum of multiple I2C values?

3DS I2C New3DS 16bit IO Expander (aka QTM)

TCA6416A 16bit IO Expander I2C Registers (I2C Bus/Device: 0:40h)

```
00h Input Port 0                    (R) (reset: var, reads as 05h on New3DS)
01h Input Port 1                    (R) (reset: var, reads as 00h on New3DS)
02h Output Port 0                   (R/W) (reset: FFh)
03h Output Port 1                   (R/W) (reset: FFh)
04h Input Polarity Inversion 0      (R/W) (reset: 00h=normal)
05h Input Polarity Inversion 1      (R/W) (reset: 00h=normal)
06h Configuration (aka Direction) 0 (R/W) (reset: FFh=input)
07h Configuration (aka Direction) 1 (R/W) (reset: FFh=input)
08h..FFh Reserved                   (-)   (FFh's)
```

Port Usage in New3DS
Unknown. The IO Expander chip is close to the top-screen connectors, as such it does probably connect to components in the top-screen unit (which might include upper camera; or reportedly-existing head-tracking related stuff).
Unknown how a 16bit IO could do head-tracking, maybe it does opposite: receive the tracking RESULT from ARM side, and then OUTPUT something to parallax barrier?
Unknown if this can be reverse-engineered by people with broken top-screen connectors. Note: Below "usually set/cleared" assumes that the connector for upper backlight is broken, unknown if an intact console behaves the same.

```
P0.0  Unknown (usually set)  ;DANGER: power-off when set to Output/Low
P0.1  Unknown (usually 0)
P0.2  Unknown (usually set)  ;DANGER: power-off when set to Output/Low
P0.3  Unknown (usually 0)
P0.4  Unknown (usually 0)
P0.5  Unknown (usually 0)
P0.6  Unknown (usually 0)
P0.7  Unknown (usually 0)
P1.0  Unknown (usually 0)
P1.1  Unknown (usually 0)
P1.2  Unknown (usually 0)
P1.3  Unknown (usually 0)
P1.4  Unknown (usually 0)
P1.5  Unknown (usually 0)
P1.6  Unknown (usually 0)
P1.7  Unknown (usually 0)
```

00h - Input Port 0 (reset: var, reads as 05h on New3DS) (R)
01h - Input Port 1 (reset: var, reads as 00h on New3DS) (R)

```
0-7  Input level (0=Low, 1=High; or vice-versa when using Polarity Inversion)
```

02h - Output Port 0 (reset: FFh) (R/W)
03h - Output Port 1 (reset: FFh) (R/W)

```
0-7  Output level (0=Low, 1=High) (no effect if direction is Input)
```

04h - Polarity Inversion 0 (reset: 00h) (R/W)
05h - Polarity Inversion 1 (reset: 00h) (R/W)

```
0-7  Polarity of Inputs (0=Normal, 1=Invert)
```

06h - Configuration 0 (reset: FFh) (R/W)
07h - Configuration 1 (reset: FFh) (R/W)

```
0-7  Direction (0=Output, 1=Input)
```

Interrupt Pin
The IO Expander can trigger interrupts on any changes on any input pins. Unknown if the Interrupt signal is connected in New3DS.

3DS I2C Other/Unused/Debug Devices

Device 7 (Bus/device 1:40h) "i2c::DEB" Debug?
Device 8 (Bus/device 1:44h) "i2c::DEB" Debug?
Unknown.

Device 9 (Bus/device 2:A6h/D6h) "i2c::HID" Debug??
Unknown.
Device Address changed from A6h to D6h in 8.0.0-18.

Device 12 (Bus/device 2:A4h) "DebugPad"

```
Register   Width   Description
00h        21      DebugPad state.
```

This is the DebugPad device, see here.

Device 14 (Bus/device 2:A0h) "eeprom?"
Used by Cfg-sysmodule via the i2c::EEP service. This is presumably EEPROM going by the service name. The Cfg-module code which loads the CCAL (nandro:/sys/{HWCAL0.dat/HWCAL1.dat}) file from NAND will load it from I2C instead, if a certain state flag is non-zero. Likewise for the function which writes CCAL to NAND. HMAC/hash verification after loading is skipped when the CCAL was loaded from I2C.

Bus/device 0:00h-0Eh - Mirrors to BPTWL for whatever reason
Bus/device 0:F0h-FEh - Mirrors to BPTWL for whatever reason
These are special broadcast/reserved device ids (defined in I2C protocol). The MCU chip apparently includes a hardware feature for decoding these IDs; and accidentally mirrors the BPTWL registers to them.

Bus/device 2:00h - Unknown, something responds here with ACK and FFh's
This is a special broadcast address (defined in I2C protocol). One (or more) of the devices on I2C Bus 2 do apparently respond to it... and perhaps even support some broadcast commands?

3DS Video

LCD Registers
These registers are used to configure the LCD screens.
3DS Video LCD Registers

Moreover, each screen is having an I2C-bus controller for further configuration.
3DS I2C LCD Screen Controllers
And, the MCU has several LCD/backlight enable flags... and a GPU flag?

GPU Memory and I/O Map
3DS GPU Memory and I/O Map
3DS GPU External Register List Summary
3DS GPU Internal Register List Summary

GPU External Registers (for memory transfers, and framebuf to LCD output)
3DS GPU External Registers - Memory Control/Status Registers
3DS GPU External Registers - Top/Bottom Screen and Framebuffer Setup
3DS GPU External Registers - Memfill and Memcopy

GPU Internal Registers (PICA200) (for drawing polygons to colorbuf/depthbuf)
3DS GPU Internal Register Overview
3DS GPU Internal Registers - Command Lists
3DS GPU Internal Registers - Finalize Interrupt registers
3DS GPU Internal Registers - Geometry Pipeline registers
3DS GPU Internal Registers - Shader registers
3DS GPU Internal Registers - Rasterizer registers
3DS GPU Internal Registers - Framebuffer registers
3DS GPU Internal Registers - Texturing registers (Generic Textures)
3DS GPU Internal Registers - Texturing registers (Procedural Texture)
3DS GPU Internal Registers - Texturing registers (Environment)
3DS GPU Internal Registers - Fragment Lighting registers
3DS GPU Internal Registers - Unknown/Unused/Undocumented Registers
3DS GPU Shader Instruction Set - Opcode Summary
3DS GPU Shader Instruction Set - Blurp
Misc notes...
3DS GPU Geometry Pipeline
3DS GPU Fragment Lighting
3DS GPU Procedural Texture Generation
3DS GPU Pitfalls
3DS GPU Primitive Engine and Shaders
There is also some 3DS GPU open source driver called citro3d, but it's mostly higher level stuff and would require a lot of reverse engineering to extract to lower level functionality from it.

Sample Code
3DS GPU Triangle Drawing Sample Code

Camera and Video Data Conversion
3DS Video CAM Registers (Camera Input)
3DS Video Y2R Registers (YUV-to-RGBA Converter)
3DS Video L2B Registers (RGB-to-RGBA Converter) (New3DS)
3DS Video MVD Registers (Movie Decoder or so?) (New3DS)
3DS Video LGY Registers (Legacy GBA/NDS Video to Framebuffer)

3DS Video LCD Registers

This seems to be an interface between GPU video output and actual LCD screens. The registers allow to control backlight brightness and to set a forced blank color. Some other registers seem to be also affect backlight

brightness (or pixel brightness?); and maybe some do affect internal voltages or internal backlight PWM timings?

The bootrom error screen initializes only a few of these registers (and leaves the others at their power-up defaults). Many of the uninitialized registers seem to have no effect on the visible picture, even when setting/clearing all bits; some of them affect the brightness of the pixels or backlight (but only if CtrlFlags.bit0=1).

General LCD Control Registers

```
  Physical  Size R/W       Name
  10202000h 4    00070007h reportedly Parallax barrier enable (uh, in 2x3bit?)
  10202004h 4    FFFFFFFFh Whatever, should be 0A390A39h
  10202008h 4    (R)       Whatever, readonly (R) (bit0=top screen mode?)
  1020200Ch 4    00010001h Video Disable Bits (bit0/16=top/bottom disable?)
  10202010h 4    00001F0Fh Whatever, usually 00000900h
  10202014h 4    00000001h Video Enable Bit (bit0)
  10202018h 1E8h (0)       Unused (0) ?
  10202200h 600h ...       LCD Configuration for Top Screen (see below)
  10202800h 200h (0)       Unused (0) ?
  10202A00h 600h ...       LCD Configuration for Bottom Screen (see below)
 (10203200h)40h  (0) uh??  reportedly LCD related, but actually DSP-mirror
```

LCD Configuration for Top/Bottom Screen

```
  Top/Bottom Addr     Size R/W
  10202200h/10202A00h 4    00000301h CtrlFlags
  10202204h/10202A04h 4    01FFFFFFh Forced Blank Color
  10202208h/10202A08h 8    (0)       Unused (0) ?
  10202210h/10202A10h 4    000003FFh ? (default=0)    ;\
  10202214h/10202A14h 4    000003FFh ? (default=3FFh) ; darker when all zero
  10202218h/10202A18h 4    000003FFh ? (default=0)    ; (if CtrlFlags.bit0=1)
  1020221Ch/10202A1Ch 4    000003FFh ? (default=3FFh) ;/
  10202220h/10202A20h 4    000001FFh ? (default=100h) ;-
  10202224h/10202A24h 4    000000FFh ? (default=0)    ;\
  10202228h/10202A28h 4    000000FFh ? (default=0)    ;
  1020222Ch/10202A2Ch 4    000000FFh ? (default=0)    ;/
  10202230h/10202A30h 4    001F000Fh ? (default=4)    ;-
  10202234h/10202A34h 4    (0)       Unused (0) ?
  10202238h/10202A38h 4    000000FFh Last same entry in 10202400h (FFh=AllSame)
  1020223Ch/10202A3Ch 4    000000FFh ? (default=FFh)
  10202240h/10202A40h 4    000003FFh Backlight ;based on register 244h.low10bit
  10202244h/10202A44h 4    FFFFF3FFh Backlight?;LSBs: backlight? MSBs=flags?
  10202248h/10202A48h 8    (0)       Unused (0) ?
  10202250h/10202A50h 4    00FFFFFFh ? (default=0)    ;\brighter if 2nd=FFFFFFh
  10202254h/10202A54h 4    00FFFFFFh ? (default=0)    ;/(if CtrlFlags.bit0=1)
  10202258h/10202A58h 8    (0)       Unused (0) ?
  10202260h/10202A60h 4    0000CCCCh ? (default=0C84h);\
  10202264h/10202A64h 4    (0)       Unused (0) ?     ;
  10202268h/10202A68h 4    0000CCCCh ? (default=840Ch); DITHER alike Y2R ?
  1020226Ch/10202A6Ch 4    (0)       Unused (0) ?     ;
  10202270h/10202A70h 4    0000CCCCh ? (default=48C0h);
  10202274h/10202A74h 4    (0)       Unused (0) ?     ;
  10202278h/10202A78h 4    0000CCCCh ? (default=C048h);
  1020227Ch/10202A7Ch 4    (0)       Unused (0) ?     ;/
  10202280h/10202A80h 24h  000000FFh ? (default=0)    ;-9 words (8th=brighter)
  102022A4h/10202AA4h 4Ch  (0)       Unused (0) ?
  102022F0h/10202AF0h 4    (-R-)     ? (R)            ;\initially 00h, but can
  102022F4h/10202AF4h 4    (-R-)     ? (R)            ; be FFh when screens
  102022F8h/10202AF8h 4    (-R-)     ? (R)            ;/are on?
  102022FCh/10202AFCh 4    (0)       Unused (0) ?
  10202300h/10202B00h 100h 00FFFFFFh New3DS only: LCD calibration array?
  10202400h/10202C00h 400h 000003FFh Backlight related array, used by bootrom
```

General Control

10202000h - Parallax barrier enable (uh, in 2x3bit?) (R/W)
```
0-2    Unknown (0..7)
3-15   Unused (0)
16-18  Unknown (0..7)
19-31  Unused (0)
```
No visible effect, at least not on bottom screen.

10202004h - Whatever, should be 0A390A39h (R/W)
```
0-15   Unknown
16-31  Unknown
```
Usually set to 0A390A39h. But there's no visible effect when set to 0 or FFFFFFFFh.

10202008h - Whatever, readonly (R)
```
0      Indicates current horizontal resolution of top screen?
1-31   Unknown or unused
```

1020200Ch - Video Disable Bits (R/W)
```
0      Unknown (maybe same as bit16, for top screen?)
1-15   Unused (0)
16     Disable Bottom Screen (0=Normal/On, 1=Screen fades out)
17-31  Unused (0)
```

10202010h - Whatever, usually 00000900h (R/W)
```
0-3    Unknown (0..0Fh)
4-7    Unused (0)
8-12   Unknown (0..1Fh)
13-31  Unused (0)
```
No visible effect. Power-up value is 00000900h. Bootrom doesn't change that setting.

10202014h - Video Enable Bit (R/W)
```
0      Video enable (0=Off/Black, 1=On/Picture)
1-31   Unused (0)
```
Bottom (and top?) screen goes black when off (though backlight is kept on).

Top/Bottom Screen

10202200h/10202A00h - CtrlFlags (R/W)
```
0      Enables custom settings when set?
1-7    Unused (0)
8-9    Unknown (0..3)
10-31  Unused (0)
```
Bit0 should be copied to 10202244h/10202A44h.bit17?
Bit0 seems to enable custom settings in 1020221xh/10202A1xh, 10202254h/10202A54h, and in the eighth word at 10202280h/10202A80h.

10202204h/10202A04h - Forced Blank Color (R/W)
```
0-7    Blank Color Red     (00h..FFh)
8-15   Blank Color Green   (00h..FFh)
16-23  Blank Color Blue    (00h..FFh)
24     Blank Color Enable  (0=Normal/Picture, 1=Force Blank Color)
25-31  Unused (0)
```
The blank color overrides the whole picture (including the Screen Border area).

10202210h/10202A10h 4 000003FFh ? (default=0) ;
10202214h/10202A14h 4 000003FFh ? (default=3FFh) ;
10202218h/10202A18h 4 000003FFh ? (default=0) ;
1020221Ch/10202A1Ch 4 000003FFh ? (default=3FFh) ;/

```
  0-9    Unknown (0..3FFh)
  10-31  Unused (0)
```
Screen gets darker when set to all zeroes (but only if CtrlFlags.bit0=1).

10202220h/10202A20h 4 000001FFh ? (default=100h) ;-
```
  0-8    Unknown (0..1FFh)
  9-31   Unused (0)
```

10202224h/10202A24h 4 000000FFh ? (default=0) ;
10202228h/10202A28h 4 000000FFh ? (default=0) ;
1020222Ch/10202A2Ch 4 000000FFh ? (default=0) ;/
```
  0-7    Unknown (0..FFh)
  8-31   Unused (0)
```

10202230h/10202A30h 4 001F000Fh ? (default=4) ;-
```
  0-3    Unknown (0..0Fh)
  4-15   Unused (0)
  16-20  Unknown (0..1Fh)
  21-31  Unused (0)
```

10202238h/10202A38h 4 000000FFh Last same entry in 10202400h/10202C00h
```
  0-7    Index of last SAME entry in 10202400h/10202C00h (0..FFh) (FFh=AllSame)
  8-31   Unused (0)
```

1020223Ch/10202A3Ch 4 000000FFh ? (default=FFh)
```
  0-7    Unknown (0..FFh)
  8-31   Unused (0)
```

10202240h/10202A40h 4 000003FFh Backlight ;based on register 244h.low10bit
```
  0-9    Backlight related (0..3FFh)
  10-31  Unused (0)
```

10202244h/10202A44h 4 FFFFF3FFh Backlight?;LSBs: backlight? MSBs=flags?
```
  0-9    Backlight related? (0..3FFh)
  10-11  Unused (0)
  12-15  Unknown (should be 0 or 6, depending on MCU?)
  16     Unknown
  17     Unknown
  18-31  More unknown bits/values...
```

10202250h/10202A50h 4 00FFFFFFh ? (default=0)
10202254h/10202A54h 4 00FFFFFFh ? (default=0)
```
  0-23   Unknown (0..FFFFFFh)
  24-31  Unused (0)
```
Two 24bit values. Screen gets brighter when setting the SECOND value to FFFFFFh (but only if CtrlFlags.bit0=1).

10202260h/10202A60h 4 0000CCCCh ? (default=0C84h)
10202268h/10202A68h 4 0000CCCCh ? (default=840Ch)
10202270h/10202A70h 4 0000CCCCh ? (default=48C0h)
10202278h/10202A78h 4 0000CCCCh ? (default=C048h)
```
  0-15   Unknown (R/W Mask CCCCh)
  16-31  Unused (0)
```
DITHER alike Y2R?

10202280h/10202A80h 24h 000000FFh ? (default=0) ;-(9 words)
```
  0-7    Unknown (0..FFh)
  8-31   Unused (0)
```

Unknown... 9 words (with 8bit used each) at 10202280h/10202A80h and up.
Screen gets brighter when setting the EIGHTH value to FFh (but only if CtrlFlags.bit0=1).

102022F0h/10202AF0h - Readonly (R)
102022F4h/10202AF4h - Readonly (R)
102022F8h/10202AF8h - Readonly (R)

```
  0-xx   Unknown (initially 00h, but can be FFh when screens are on?)  (R)
  xx-31  Unused (0)
```

10202300h/10202B00h 100h 00FFFFFFh New3DS only: LCD calibration? (40 words)

```
  0-23   Unknown (0..FFFFFFh) (initially random)
  24-31  Unused (0)
```

40h words (100h bytes) at 10202300h/10202B00h and up.
Reportedly initialized from nand:/ro/sys/HWCAL0.dat offset 77Ch.
On Old3DS: Unused (0)
On New3DS: R/W (even in Old3DS mode)
Bootrom doesn't init these values (so they are apparently initially disabled somewhere).

10202400h/10202C00h 400h 000003FFh Whatever array, used by bootrom!!!

```
  0-9    Unknown (0..3FFh) (initially random)
  10-31  Unused (0)
```

100h words (400h bytes) at 10202400h/10202C00h and up.
The bootrom error screen has some complicated function that initializes this array with scaled/interpolated values, somehow related to backlight level and screen type; although, with the used parameters... it seems to be just settings all 100h table entries to the same value.
Alongsides, the number of leading entries with SAME value should be stored in register 10202238h/10202A38h (ie. store N-1 if the first N table entries are all same).

3DS GPU Memory and I/O Map

GPU Overall Memory and I/O Map
All registers at 10202xxxh, 10400xxxh, 10401xxxh can be read/written by ARM11 opcodes, the 10401xxxh registers can be also written via command numbers 0000h..03FFh in PICA command lists.

```
  10202000h 18h     LCD Config/flags                                ;\
  10202018h 1E8h    Unused (0) ?                                    ;
  10202200h 600h    LCD Configuration for Top Screen                ; LCD
  10202800h 200h    Unused (0) ?                                    ;
  10202A00h 600h    LCD Configuration for Bottom Screen             ;/
  10400000h 10h     GPU Memory Control/Status Registers (part 1)    ;\
  10400010h 10h     DMA Memory Fill 0 DMA "PSC0"                    ;
  10400020h 10h     DMA Memory Fill 1 DMA "PSC1"                    ;
  10400030h D0h     GPU Memory Control/Status Registers (part 2)    ; External
  10400100h 300h    Unused (mirrors of above 100h bytes)            ;
  10400400h 100h    LCD Framebuffer Setup "PDC0" (top screen)       ;
  10400500h 100h    LCD Framebuffer Setup "PDC1" (bottom screen)    ;
  10400600h 200h    Unknown/DANGER (hangs when reading?)            ;
  10400800h 400h    Unknown/Unused (data abort)                     ;
  10400C00h 100h    DMA Memory Copy DMA Transfer Engine "DMA"       ;
  10400D00h 300h    Unused (mirrors of above 100h bytes)            ;/
  10401000h 1000h   Internal Command Registers PICA(0000h..03FFh)   ;-Internal
  10402000h 2000h   Unused (data abort)
  10404000h 1C000h  Write-only mirrors of above 4000h bytes (read=data abort)
  18000000h 300000h VRAM area, usually VRAM_A (3Mbyte)
  18300000h 300000h VRAM area, usually VRAM_B (3Mbyte)
  18600000h 600000h VRAM area, usually empty/zerofilled (6Mbyte)
```

Apart from VRAM, the GPU can also access other memory like FCRAM, AXI, QTM (but that's most probably slower; and may be problematic when using virtual memory allocation on ARM11 side).

3DS GPU External Register List Summary

GPU Memory Control/Status Registers - Misc

```
10400000h  GPU_FIXED_00010002h (always 00010002h, maybe ChipID/Version?) (R)
10400004h  GPU_VRAM_CNT (reset=0, bootrom=300h) (R/W)
10400008h  GPU_UNKNOWN_2BIT (R/W)
1040000Ch  Unused (0)
```

GPU Memory Fill Unit 0 and 1

```
10400010h  GPU_MEMFILL_DST_ADDR0     ;\
10400014h  GPU_MEMFILL_DST_END0      ; GPU_MEMFILL_0
10400018h  GPU_MEMFILL_DATA0         ;
1040001Ch  GPU_MEMFILL_CNT0          ;/
10400020h  GPU_MEMFILL_DST_ADDR1     ;\
10400024h  GPU_MEMFILL_DST_END1      ; GPU_MEMFILL_1
10400028h  GPU_MEMFILL_DATA1         ;
1040002Ch  GPU_MEMFILL_CNT1          ;/
```

GPU Memory Control/Status Registers - Misc

```
10400030h  GPU_VRAM_POWER (R/W)
10400034h  GPU_STAT_IRQ_FLAGS (R)
10400038h  GPU_STAT_SOMETHING (R)
1040003Ch  GPU_MAKE_SOMETHING (R/W)
10400040h  GPU_STAT_BACKLIGHT_OR_SO (R)
10400044h  GPU_UNKNOWN_CAN_BE_7F80h (R)
10400048h  GPU_UNKNOWN_32BIT (reset=0) (R/W)
1040004Ch  Unused (0)
10400050h  GPU_TIMING_CNT1 (R/W=FFFFFFFFh) (res=1111EF00h) ;init=22221200h
10400054h  GPU_TIMING_CNT2 (R/W=FFFFFFFFh) (res=00000112h) ;init=00000FF2h
10400058h  GPU_STAT_BUSY_FLAGS (R)
1040005Ch  GPU_MAKE_WHATEVER_MESS (R/W)
10400060h  GPU_STAT_WHATEVER_MESS (R)
10400064h  GPU_UNKNOWN_1BIT (R/W=00000001h) (res=0)
10400068h  GPU_UNKNOWN_INITIALLY_00A80000h (res=00A80000h) (R/W)
1040006Ch  Unused (0)
```

GPU Memory Control/Status Registers - Read-only memory traffic counters...

```
10400070h  GPU_STAT_TOTAL_NON_VRAM_READS      ;\Non-VRAM (eg. AXI,QTM,FCRAM)
10400074h  GPU_STAT_TOTAL_NON_VRAM_WRITES     ;/
10400078h  GPU_STAT_TOTAL_VRAM_A_READS        ;\1st 3Mbyte VRAM block
1040007Ch  GPU_STAT_TOTAL_VRAM_A_WRITES       ;/
10400080h  GPU_STAT_TOTAL_VRAM_B_READS        ;\2nd 3Mbyte VRAM block
10400084h  GPU_STAT_TOTAL_VRAM_B_WRITES       ;/
10400088h  GPU_STAT_POLYGON_ARRAY_READS       ;-ATTR_BUF, INDEX_LIST
1040008Ch  GPU_STAT_POLYGON_TEXTURE_READS     ;-cache misses only
10400090h  GPU_STAT_POLYGON_DEPTHBUFFER_READS
10400094h  GPU_STAT_POLYGON_DEPTHBUFFER_WRITES
10400098h  GPU_STAT_POLYGON_COLORBUFFER_READS
1040009Ch  GPU_STAT_POLYGON_COLORBUFFER_WRITES
104000A0h  GPU_STAT_LCD_UPPER_SCREEN_READS    ;\for both left+right buffer
104000A4h  GPU_STAT_LCD_LOWER_SCREEN_READS    ;/
104000A8h  GPU_STAT_MEMCOPY_SRC_READS         ;\GPU_MEMCOPY
104000ACh  GPU_STAT_MEMCOPY_DST_WRITES        ;/
104000B0h  GPU_STAT_MEMFILL_0_DST_WRITES      ;\GPU_MEMFILL 0/1
104000B4h  GPU_STAT_MEMFILL_1_DST_WRITES      ;/
104000B8h  GPU_STAT_CPU_READS_FROM_VRAM_A_B   ;\counted by GPU because the
104000BCh  GPU_STAT_CPU_WRITES_TO_VRAM_A_B    ;/GPU must forward vram to cpu
```

GPU Memory Control/Status Registers - Misc

```
104000C0h  GPU_BACKLIGHT_OR_SO_1 (reset=20000000h) (R/W)
104000C4h  GPU_BASE_ADDR_VRAM_A (reset=18000000h) (R/W)
104000C8h  GPU_BASE_ADDR_VRAM_B (reset=18300000h) (R/W)
104000CCh  GPU_BACKLIGHT_OR_SO_2 (reset=20000000h) (R/W)
104000D0h  GPU_UNKNOWN_4BIT (reset=0) (R/W)
```

```
  104000D4h  Unused (2Ch bytes, zerofilled)
  10400100h  Unused (300h bytes, mirrors of above 100h bytes)
```
GPU Display Contollers
```
  10400400h  LCD Framebuffer Setup "PDC0" (100h bytes) (top screen)
  10400500h  LCD Framebuffer Setup "PDC1" (100h bytes) (bottom screen)
  10400600h  Unused? (200h bytes, Unknown/DANGER, hangs when reading?)
  10400800h  Unused? (400h bytes, Unknown/Unused, data abort)
```
GPU Memory Copy Unit
```
  10400C00h  GPU_MEMCOPY_SRC_ADDR - Memcopy Input physical address (R/W)
  10400C04h  GPU_MEMCOPY_DST_ADDR - Memcopy Output physical address (R/W)
  10400C08h  GPU_MEMCOPY_DISPLAY_SIZE - DisplayTransfer width/height (R/W)
  10400C0Ch  GPU_MEMCOPY_DISPLAY_GAP - DisplayTransfer Input height+gap (R/W)
  10400C10h  GPU_MEMCOPY_FLAGS - Memcopy Transfer flags (R/W)
  10400C14h  GPU_MEMCOPY_UNKNOWN_21BIT - Memcopy (R/W)
  10400C18h  GPU_MEMCOPY_CNT - Memcopy Start/Busy (R/W)
  10400C1Ch  GPU_MEMCOPY_REMAIN_IRQ - Memcopy Remain IRQ (R/W)
  10400C20h  GPU_MEMCOPY_TEX_SIZE - TextureCopy total amount
  10400C24h  GPU_MEMCOPY_TEX_SRC_WIDTH - TextureCopy input line width/gap
  10400C28h  GPU_MEMCOPY_TEX_DST_WIDTH - TextureCopy output line width/gap
  10400C2Ch  GPU_MEMCOPY_UNKNOWN_FLAG - Memcopy Unknown (R/W)
  10400C30h  Unused (D0h bytes, zerofilled)
  10400D00h  Unused (300h bytes, mirrors of above 100h bytes)
```

3DS GPU Internal Register List Summary

3DS GPU Internal Registers - Finalize Interrupt registers
```
  10401000h+i   - PICA(N/A)       - GPUREG_IRQ_ACK(0..63) (R/W)
  10401040h+i   - PICA(0010h+i/4) - GPUREG_IRQ_REQ(0..63) (R/W)
  10401080h+i   - PICA(N/A)       - GPUREG_IRQ_CMP(0..63) (R/W)
  104010C0h     - PICA(N/A)       - GPUREG_IRQ_MASK_LOW (R/W)
  104010C4h     - PICA(N/A)       - GPUREG_IRQ_MASK_HIGH (R/W)
  104010C8h     - PICA(N/A)       - GPUREG_IRQ_STAT_LOW (R)
  104010CCh     - PICA(N/A)       - GPUREG_IRQ_STAT_HIGH (R)
  104010D0h     - PICA(N/A)       - GPUREG_IRQ_AUTOSTOP (R/W)
  104010D4h     - PICA(N/A)       - GPUREG_FIXED_00010002h (R)
```
3DS GPU Internal Registers - Rasterizer registers
```
  10401100h     - PICA(0040h)   - GPUREG_FACECULLING_CONFIG (R/W)
  10401104h     - PICA(0041h)   - GPUREG_VIEWPORT_V_SCALE (R/W)
  10401108h     - PICA(0042h)   - GPUREG_VIEWPORT_V_STEP (R/W)
  1040110Ch     - PICA(0043h)   - GPUREG_VIEWPORT_H_SCALE (R/W)
  10401110h     - PICA(0044h)   - GPUREG_VIEWPORT_H_STEP (R/W)
  10401114h     - PICA(0045h)   - GPUREG_undoc_10401114h (R/W=00FFFFFFh)
  10401118h     - PICA(0046h)   - GPUREG_undoc_10401118h (R/W=00FFFFFFh)
  1040111Ch     - PICA(0047h)   - GPUREG_FRAGOP_CLIP (R/W)
  10401120h+i*4 - PICA(0048h+i) - GPUREG_FRAGOP_CLIP_DATAi (i=0..3) (R/W)
  10401130h     - PICA(004Ch)   - GPUREG_undoc_10401130h (R/W=00000001h)
  10401134h     - PICA(004Dh)   - GPUREG_DEPTHMAP_SCALE (R/W)
  10401138h     - PICA(004Eh)   - GPUREG_DEPTHMAP_OFFSET (R/W)
  1040113Ch     - PICA(004Fh)   - GPUREG_SH_OUTMAP_TOTAL (R/W)
  10401140h+i*4 - PICA(0050h+i) - GPUREG_SH_OUTMAP_Oi (i=0..6) (R/W)
  10401160h     - PICA(0058h)   - GPUREG_undoc_10401160h (R/W=00000101h)
  10401164h     - PICA(0059h)   - GPUREG_undoc_10401164h (R/W=00000001h)
  10401168h     - PICA(005Ah)   - GPUREG_STAT_NUM_VERTICES_RECEIVED (R)
  1040116Ch     - PICA(005Bh)   - GPUREG_STAT_NUM_TRIANGLES_RECEIVED (R)
  10401170h     - PICA(005Ch)   - GPUREG_STAT_NUM_TRIANGLES_DISPLAYED (R)
  10401180h     - PICA(0060h)   - GPUREG_undoc_10401180h (R/W=00000301h)
  10401184h     - PICA(0061h)   - GPUREG_EARLYDEPTH_FUNC (R/W)
  10401188h     - PICA(0062h)   - GPUREG_EARLYDEPTH_TEST1 (R/W)
  1040118Ch     - PICA(0063h)   - GPUREG_EARLYDEPTH_CLEAR (W)
  10401190h     - PICA(0064h)   - GPUREG_SH_OUTATTR_MODE (R/W)
  10401194h     - PICA(0065h)   - GPUREG_SCISSORTEST_MODE (R/W)
  10401198h     - PICA(0066h)   - GPUREG_SCISSORTEST_POS1 (R/W)
```

```
1040119Ch     - PICA(0067h)   - GPUREG_SCISSORTEST_POS2 (R/W)
104011A0h     - PICA(0068h)   - GPUREG_VIEWPORT_XY (R/W)
104011A4h     - PICA(0069h)   - GPUREG_undoc_104011A4h (R/W=FFFF0001h)
104011A8h     - PICA(006Ah)   - GPUREG_EARLYDEPTH_DATA (R/W)
104011ACh     - PICA(006Bh)   - GPUREG_undoc_104011ACh (R/W=00000FFFh)
104011B0h     - PICA(006Ch)   - GPUREG_undoc_104011B0h (hangs when reading)
104011B4h     - PICA(006Dh)   - GPUREG_DEPTHMAP_ENABLE (R/W)
104011B8h     - PICA(006Eh)   - GPUREG_RENDERBUFFER_DIM_1 (R/W)
104011BCh     - PICA(006Fh)   - GPUREG_SH_OUTATTR_CLOCK (R/W)
```

3DS GPU Internal Registers - Texturing registers (Generic Textures)

```
10401200h     - PICA(0080h)   - GPUREG_TEXUNIT_CONFIG (R/W)        ;-CONFIG
10401204h     - PICA(0081h)   - GPUREG_TEXUNIT0_BORDER_COLOR (R/W) ;\
10401208h     - PICA(0082h)   - GPUREG_TEXUNIT0_DIM (R/W)          ;
1040120Ch     - PICA(0083h)   - GPUREG_TEXUNIT0_PARAM (R/W)        ; UNIT0
10401210h     - PICA(0084h)   - GPUREG_TEXUNIT0_LOD (R/W)          ;
10401214h     - PICA(0085h)   - GPUREG_TEXUNIT0_ADDR1 (R/W)        ;
10401218h     - PICA(0086h)   - GPUREG_TEXUNIT0_ADDR2 (R/W)        ;
1040121Ch     - PICA(0087h)   - GPUREG_TEXUNIT0_ADDR3 (R/W)        ;
10401220h     - PICA(0088h)   - GPUREG_TEXUNIT0_ADDR4 (R/W)        ;
10401224h     - PICA(0089h)   - GPUREG_TEXUNIT0_ADDR5 (R/W)        ;
10401228h     - PICA(008Ah)   - GPUREG_TEXUNIT0_ADDR6 (R/W)        ;
1040122Ch     - PICA(008Bh)   - GPUREG_TEXUNIT0_SHADOW (R/W)       ;
10401230h     - PICA(008Ch)   - GPUREG_undoc_10401230h (R/W=FFFF00FFh)
10401234h     - PICA(008Dh)   - GPUREG_undoc_10401234h (R/W=000000FFh)
10401238h     - PICA(008Eh)   - GPUREG_TEXUNIT0_TYPE (R/W)         ;/
1040123Ch     - PICA(008Fh)   - GPUREG_LIGHTING_ENABLE (R/W)       ;-
10401244h     - PICA(0091h)   - GPUREG_TEXUNIT1_BORDER_COLOR (R/W) ;\
10401248h     - PICA(0092h)   - GPUREG_TEXUNIT1_DIM (R/W)          ;
1040124Ch     - PICA(0093h)   - GPUREG_TEXUNIT1_PARAM (R/W)        ; UNIT1
10401250h     - PICA(0094h)   - GPUREG_TEXUNIT1_LOD (R/W)          ;
10401254h     - PICA(0095h)   - GPUREG_TEXUNIT1_ADDR (R/W)         ;
10401258h     - PICA(0096h)   - GPUREG_TEXUNIT1_TYPE (R/W)         ;/
10401264h     - PICA(0099h)   - GPUREG_TEXUNIT2_BORDER_COLOR (R/W) ;\
10401268h     - PICA(009Ah)   - GPUREG_TEXUNIT2_DIM (R/W)          ;
1040126Ch     - PICA(009Bh)   - GPUREG_TEXUNIT2_PARAM (R/W)        ; UNIT2
10401270h     - PICA(009Ch)   - GPUREG_TEXUNIT2_LOD (R/W)          ;
10401274h     - PICA(009Dh)   - GPUREG_TEXUNIT2_ADDR (R/W)         ;
10401278h     - PICA(009Eh)   - GPUREG_TEXUNIT2_TYPE (R/W)         ;/
```

3DS GPU Internal Registers - Texturing registers (Procedural Texture)

```
104012A0h     - PICA(00A8h)   - GPUREG_TEXUNIT3_PROCTEX0 (R/W)
104012A4h     - PICA(00A9h)   - GPUREG_TEXUNIT3_PROCTEX1 (R/W)
104012A8h     - PICA(00AAh)   - GPUREG_TEXUNIT3_PROCTEX2 (R/W)
104012ACh     - PICA(00ABh)   - GPUREG_TEXUNIT3_PROCTEX3 (R/W)
104012B0h     - PICA(00ACh)   - GPUREG_TEXUNIT3_PROCTEX4 (R/W)
104012B4h     - PICA(00ADh)   - GPUREG_TEXUNIT3_PROCTEX5_LOW (R/W)
104012B8h     - PICA(00AEh)   - GPUREG_TEXUNIT3_PROCTEX5_HIGH (R/W)
104012BCh     - PICA(00AFh)   - GPUREG_PROCTEX_LUT_INDEX (R/W)
104012C0h+i*4 - PICA(00B0h+i) - GPUREG_PROCTEX_LUT_DATA(0..7) (R/W)
```

3DS GPU Internal Registers - Texturing registers (Environment)

```
10401300h     - PICA(00C0h)   - GPUREG_TEXENV0_SOURCE (R/W)       ;\
10401304h     - PICA(00C1h)   - GPUREG_TEXENV0_OPERAND (R/W)      ;
10401308h     - PICA(00C2h)   - GPUREG_TEXENV0_COMBINER (R/W)     ; ENV0
1040130Ch     - PICA(00C3h)   - GPUREG_TEXENV0_COLOR (R/W)        ;
10401310h     - PICA(00C4h)   - GPUREG_TEXENV0_SCALE (R/W)        ;/
1040131Ch     - PICA(00C7h)   - GPUREG_undoc_1040131Ch (R/W=00000007h)
10401320h     - PICA(00C8h)   - GPUREG_TEXENV1_SOURCE (R/W)       ;\
10401324h     - PICA(00C9h)   - GPUREG_TEXENV1_OPERAND (R/W)      ;
10401328h     - PICA(00CAh)   - GPUREG_TEXENV1_COMBINER (R/W)     ; ENV1
1040132Ch     - PICA(00CBh)   - GPUREG_TEXENV1_COLOR (R/W)        ;
10401330h     - PICA(00CCh)   - GPUREG_TEXENV1_SCALE (R/W)        ;/
10401340h     - PICA(00D0h)   - GPUREG_TEXENV2_SOURCE (R/W)       ;\
10401344h     - PICA(00D1h)   - GPUREG_TEXENV2_OPERAND (R/W)      ;
10401348h     - PICA(00D2h)   - GPUREG_TEXENV2_COMBINER (R/W)     ; ENV2
1040134Ch     - PICA(00D3h)   - GPUREG_TEXENV2_COLOR (R/W)        ;
```

```
10401350h     - PICA(00D4h)   - GPUREG_TEXENV2_SCALE (R/W)        ;/
10401360h     - PICA(00D8h)   - GPUREG_TEXENV3_SOURCE (R/W)       ;\
10401364h     - PICA(00D9h)   - GPUREG_TEXENV3_OPERAND (R/W)      ;
10401368h     - PICA(00DAh)   - GPUREG_TEXENV3_COMBINER (R/W)     ; ENV3
1040136Ch     - PICA(00DBh)   - GPUREG_TEXENV3_COLOR (R/W)        ;
10401370h     - PICA(00DCh)   - GPUREG_TEXENV3_SCALE (R/W)        ;/
10401380h     - PICA(00E0h)   - GPUREG_TEXENV_UPDATE_BUFFER (R/W)
10401384h     - PICA(00E1h)   - GPUREG_FOG_COLOR (R/W)
10401388h     - PICA(00E2h)   - GPUREG_undoc_10401388h (R/W=0000FFFFh)
1040138Ch     - PICA(00E3h)   - GPUREG_undoc_1040138Ch (R/W=0000FFFFh)
10401390h     - PICA(00E4h)   - GPUREG_GAS_ATTENUATION (R/W)
10401394h     - PICA(00E5h)   - GPUREG_GAS_ACCMAX (R/W)
10401398h     - PICA(00E6h)   - GPUREG_FOG_LUT_INDEX (R/W)
104013A0h+i*4 - PICA(00E8h+i) - GPUREG_FOG_LUT_DATA(0..7) (R/W)
104013C0h     - PICA(00F0h)   - GPUREG_TEXENV4_SOURCE (R/W)       ;\
104013C4h     - PICA(00F1h)   - GPUREG_TEXENV4_OPERAND (R/W)      ;
104013C8h     - PICA(00F2h)   - GPUREG_TEXENV4_COMBINER (R/W)     ; ENV4
104013CCh     - PICA(00F3h)   - GPUREG_TEXENV4_COLOR (R/W)        ;
104013D0h     - PICA(00F4h)   - GPUREG_TEXENV4_SCALE (R/W)        ;/
104013E0h     - PICA(00F8h)   - GPUREG_TEXENV5_SOURCE (R/W)       ;\
104013E4h     - PICA(00F9h)   - GPUREG_TEXENV5_OPERAND (R/W)      ;
104013E8h     - PICA(00FAh)   - GPUREG_TEXENV5_COMBINER (R/W)     ; ENV5
104013ECh     - PICA(00FBh)   - GPUREG_TEXENV5_COLOR (R/W)        ;
104013F0h     - PICA(00FCh)   - GPUREG_TEXENV5_SCALE (R/W)        ;/
104013F4h     - PICA(00FDh)   - GPUREG_TEXENV_BUFFER_COLOR (R/W)  ;-
```

3DS GPU Internal Registers - Framebuffer registers

```
10401400h     - PICA(0100h)   - GPUREG_COLOR_OPERATION (R/W)
10401404h     - PICA(0101h)   - GPUREG_BLEND_FUNC (R/W)
10401408h     - PICA(0102h)   - GPUREG_LOGIC_OP (R/W)
1040140Ch     - PICA(0103h)   - GPUREG_BLEND_COLOR (R/W)
10401410h     - PICA(0104h)   - GPUREG_FRAGOP_ALPHA_TEST (R/W)
10401414h     - PICA(0105h)   - GPUREG_STENCIL_TEST (R/W)
10401418h     - PICA(0106h)   - GPUREG_STENCIL_OP (R/W)
1040141Ch     - PICA(0107h)   - GPUREG_DEPTH_COLOR_MASK (R/W)
10401434h     - PICA(010Dh)   - GPUREG_undoc_10401434h (R/W=00000001h)
10401438h     - PICA(010Eh)   - GPUREG_undoc_10401438h (R/W=FFFFFFFFh)
1040143Ch     - PICA(010Fh)   - GPUREG_undoc_1040143Ch (R/W=FFFFFFFFh)
10401440h     - PICA(0110h)   - GPUREG_RENDERBUFFER_INVALIDATE (forget) (W)
10401444h     - PICA(0111h)   - GPUREG_RENDERBUFFER_FLUSH (writeback) (W)
10401448h     - PICA(0112h)   - GPUREG_COLORBUFFER_READING (R/W)
1040144Ch     - PICA(0113h)   - GPUREG_COLORBUFFER_WRITING (R/W)
10401450h     - PICA(0114h)   - GPUREG_DEPTHBUFFER_READING (R/W)
10401454h     - PICA(0115h)   - GPUREG_DEPTHBUFFER_WRITING (R/W)
10401458h     - PICA(0116h)   - GPUREG_DEPTHBUFFER_FORMAT (R/W)
1040145Ch     - PICA(0117h)   - GPUREG_COLORBUFFER_FORMAT (R/W)
10401460h     - PICA(0118h)   - GPUREG_EARLYDEPTH_TEST2 (R/W)
10401464h     - PICA(0119h)   - GPUREG_undoc_10401464h (R/W=FFFFFFFFh)
10401468h     - PICA(011Ah)   - GPUREG_undoc_10401468h (R/W=FFFFFFFFh)
1040146Ch     - PICA(011Bh)   - GPUREG_RENDERBUFFER_BLOCK32 (R/W)
10401470h     - PICA(011Ch)   - GPUREG_DEPTHBUFFER_LOC (R/W)
10401474h     - PICA(011Dh)   - GPUREG_COLORBUFFER_LOC (R/W)
10401478h     - PICA(011Eh)   - GPUREG_RENDERBUFFER_DIM_0 (R/W)
1040147Ch     - PICA(011Fh)   - GPUREG_undoc_1040147Ch (R/W=7FFFFFFFh)
10401480h     - PICA(0120h)   - GPUREG_GAS_LIGHT_XY - Planar Shading (R/W)
10401484h     - PICA(0121h)   - GPUREG_GAS_LIGHT_Z - View Shading (R/W)
10401488h     - PICA(0122h)   - GPUREG_GAS_LIGHT_Z_COLOR (R/W)
1040148Ch     - PICA(0123h)   - GPUREG_GAS_LUT_INDEX (W)
10401490h     - PICA(0124h)   - GPUREG_GAS_LUT_DATA (R/W)
10401494h     - PICA(0125h)   - GPUREG_undoc_10401494h (R/W=0000FFFFh)
10401498h     - PICA(0126h)   - GPUREG_GAS_DELTAZ_DEPTH (R/W)
104014C0h     - PICA(0130h)   - GPUREG_FRAGOP_SHADOW (R/W)
104014FCh     - PICA(013Fh)   - GPUREG_undoc_104014FCh (R/W=0000000Fh)
```

3DS GPU Internal Registers - Fragment Lighting registers

```
10401500h+i*40h - PICA(0140h+10h*(0..7)) - GPUREG_LIGHTi_SPECULAR0 (R/W)
```

```
10401504h+i*40h - PICA(0141h+10h*(0..7)) - GPUREG_LIGHTi_SPECULAR1 (R/W)
10401508h+i*40h - PICA(0142h+10h*(0..7)) - GPUREG_LIGHTi_DIFFUSE (R/W)
1040150Ch+i*40h - PICA(0143h+10h*(0..7)) - GPUREG_LIGHTi_AMBIENT (R/W)
10401510h+i*40h - PICA(0144h+10h*(0..7)) - GPUREG_LIGHTi_VECTOR_LOW (R/W)
10401514h+i*40h - PICA(0145h+10h*(0..7)) - GPUREG_LIGHTi_VECTOR_HIGH (R/W)
10401518h+i*40h - PICA(0146h+10h*(0..7)) - GPUREG_LIGHTi_SPOTDIR_LOW (R/W)
1040151Ch+i*40h - PICA(0147h+10h*(0..7)) - GPUREG_LIGHTi_SPOTDIR_HIGH (R/W)
10401524h+i*40h - PICA(0149h+10h*(0..7)) - GPUREG_LIGHTi_CONFIG (R/W)
10401528h+i*40h - PICA(014Ah+10h*(0..7)) - GPUREG_LIGHTi_ATTENUATION_BIAS
1040152Ch+i*40h - PICA(014Bh+10h*(0..7)) - GPUREG_LIGHTi_ATTENUATION_SCALE
10401700h       - PICA(01C0h)   - GPUREG_LIGHTING_AMBIENT (R/W)
10401708h       - PICA(01C2h)   - GPUREG_LIGHTING_NUM_LIGHTS (R/W)
1040170Ch       - PICA(01C3h)   - GPUREG_LIGHTING_CONFIG0 (R/W)
10401710h       - PICA(01C4h)   - GPUREG_LIGHTING_CONFIG1 (R/W)
10401714h       - PICA(01C5h)   - GPUREG_LIGHTING_LUT_INDEX (R/W)
10401718h       - PICA(01C6h)   - GPUREG_LIGHTING_DISABLE (R/W)
10401720h+i*4   - PICA(01C8h+i) - GPUREG_LIGHTING_LUT_DATA(0..7) (R/W)
10401740h       - PICA(01D0h)   - GPUREG_LIGHTING_LUTINPUT_ABS (R/W)
10401744h       - PICA(01D1h)   - GPUREG_LIGHTING_LUTINPUT_SELECT (R/W)
10401748h       - PICA(01D2h)   - GPUREG_LIGHTING_LUTINPUT_SCALE (R/W)
10401764h       - PICA(01D9h)   - GPUREG_LIGHTING_LIGHT_PERMUTATION (R/W)
```

3DS GPU Internal Registers - Geometry Pipeline registers

```
10401800h       - PICA(0200h)   - GPUREG_ATTR_BUF_BASE (R/W)
10401804h       - PICA(0201h)   - GPUREG_ATTR_BUF_FORMAT_LOW (R/W)
10401808h       - PICA(0202h)   - GPUREG_ATTR_BUF_FORMAT_HIGH (R/W)
1040180Ch+i*0Ch - PICA(0203h+3*(0..11)) - GPUREG_ATTR_BUFi_OFFSET (R/W)
10401810h+i*0Ch - PICA(0204h+3*(0..11)) - GPUREG_ATTR_BUFi_CONFIG_LOW (R/W)
10401814h+i*0Ch - PICA(0205h+3*(0..11)) - GPUREG_ATTR_BUFi_CONFIG_HIGH (R/W)
1040189Ch       - PICA(0227h)   - GPUREG_ATTR_BUF_INDEX_LIST (R/W)
104018A0h       - PICA(0228h)   - GPUREG_ATTR_BUF_NUMVERTICES (R/W)
104018A4h       - PICA(0229h)   - GPUREG_GEOSTAGE_CONFIG (R/W)
104018A8h       - PICA(022Ah)   - GPUREG_ATTR_BUF_FIRST_INDEX (R/W)
104018B4h       - PICA(022Dh)   - GPUREG_POST_VERTEX_CACHE_NUM (R/W)
104018B8h       - PICA(022Eh)   - GPUREG_ATTR_BUF_DRAWARRAYS (W)
104018BCh       - PICA(022Fh)   - GPUREG_ATTR_BUF_DRAWELEMENTS (W)
104018C4h       - PICA(0231h)   - GPUREG_VTX_FUNC (W)
104018C8h       - PICA(0232h)   - GPUREG_FIXEDATTRIB_INDEX (W)
104018CCh+i*4   - PICA(0233h+i) - GPUREG_FIXEDATTRIB_DATA(0..2) (W)
104018E0h       - PICA(0238h)   - GPUREG_CMDBUF_SIZE0 (R/W)
104018E4h       - PICA(0239h)   - GPUREG_CMDBUF_SIZE1 (R/W)
104018E8h       - PICA(023Ah)   - GPUREG_CMDBUF_ADDR0 (aka entrypoint) (R/W)
104018ECh       - PICA(023Bh)   - GPUREG_CMDBUF_ADDR1 (aka entrypoint) (R/W)
104018F0h       - PICA(023Ch)   - GPUREG_CMDBUF_JUMP0 (jump to ADDR0) (W)
104018F4h       - PICA(023Dh)   - GPUREG_CMDBUF_JUMP1 (jump to ADDR1) (W)
10401908h       - PICA(0242h)   - GPUREG_VSH_NUM_ATTR (R/W)
1040190Ch       - PICA(0243h)   - GPUREG_undoc_1040190Ch (R/W=00000037h)
10401910h       - PICA(0244h)   - GPUREG_VSH_COM_MODE (R/W)
10401914h       - PICA(0245h)   - GPUREG_START_DRAW_FUNC0 (R/W)
10401928h       - PICA(024Ah)   - GPUREG_VSH_OUTMAP_TOTAL1 (R/W)
10401944h       - PICA(0251h)   - GPUREG_VSH_OUTMAP_TOTAL2 (R/W)
10401948h       - PICA(0252h)   - GPUREG_GSH_MISC0 (R/W)
1040194Ch       - PICA(0253h)   - GPUREG_GEOSTAGE_CONFIG2 (R/W)
10401950h       - PICA(0254h)   - GPUREG_GSH_MISC1 (R/W)
10401954h       - PICA(0255h)   - GPUREG_undoc_10401954h (R/W=00000001h)
10401978h       - PICA(025Eh)   - GPUREG_PRIMITIVE_CONFIG (R/W)
1040197Ch       - PICA(025Fh)   - GPUREG_RESTART_PRIMITIVE (R/W)
```

3DS GPU Internal Registers - Shader 0 (Geometry Shader)

```
10401A00h       - PICA(0280h)   - GPUREG_GSH_BOOLUNIFORM (R/W)
10401A04h+i*4   - PICA(0281h+i) - GPUREG_GSH_INTUNIFORM_I0..I3 (R/W)
10401A24h       - PICA(0289h)   - GPUREG_GSH_INPUTBUFFER_CONFIG (R/W)
10401A28h       - PICA(028Ah)   - GPUREG_GSH_ENTRYPOINT (R/W)
10401A2Ch       - PICA(028Bh)   - GPUREG_GSH_ATTR_PERMUTATION_LOW (R/W)
10401A30h       - PICA(028Ch)   - GPUREG_GSH_ATTR_PERMUTATION_HIGH (R/W)
10401A34h       - PICA(028Dh)   - GPUREG_GSH_OUTMAP_MASK (R/W)
```

```
10401A3Ch       - PICA(028Fh)   - GPUREG_GSH_CODETRANSFER_END (W)
10401A40h       - PICA(0290h)   - GPUREG_GSH_FLOATUNIFORM_INDEX (W)
10401A44h+i*4   - PICA(0291h+i) - GPUREG_GSH_FLOATUNIFORM_DATA(0..7) (W)
10401A6Ch       - PICA(029Bh)   - GPUREG_GSH_CODETRANSFER_INDEX (W)
10401A70h+i*4   - PICA(029Ch+i) - GPUREG_GSH_CODETRANSFER_DATA(0..7) (W)
10401A94h       - PICA(02A5h)   - GPUREG_GSH_OPDESCS_INDEX (W)
10401A98h+i*4   - PICA(02A6h+i) - GPUREG_GSH_OPDESCS_DATA(0..7) (W)
```

3DS GPU Internal Registers - Shader 1 (Vertex Shader)

```
10401AC0h       - PICA(02B0h)   - GPUREG_VSH_BOOLUNIFORM (R/W)
10401AC4h+i*4   - PICA(02B1h+i) - GPUREG_VSH_INTUNIFORM_I0..I3 (R/W)
10401AE4h       - PICA(02B9h)   - GPUREG_VSH_INPUTBUFFER_CONFIG (R/W)
10401AE8h       - PICA(02BAh)   - GPUREG_VSH_ENTRYPOINT (R/W)
10401AECh       - PICA(02BBh)   - GPUREG_VSH_ATTR_PERMUTATION_LOW (R/W)
10401AF0h       - PICA(02BCh)   - GPUREG_VSH_ATTR_PERMUTATION_HIGH (R/W)
10401AF4h       - PICA(02BDh)   - GPUREG_VSH_OUTMAP_MASK (R/W)
10401AFCh       - PICA(02BFh)   - GPUREG_VSH_CODETRANSFER_END (W)
10401B00h       - PICA(02C0h)   - GPUREG_VSH_FLOATUNIFORM_INDEX (W)
10401B04h+i*4   - PICA(02C1h+i) - GPUREG_VSH_FLOATUNIFORM_DATA(0..7) (W)
10401B2Ch       - PICA(02CBh)   - GPUREG_VSH_CODETRANSFER_INDEX (W)
10401B30h+i*4   - PICA(02CCh+i) - GPUREG_VSH_CODETRANSFER_DATA(0..7) (W)
10401B54h       - PICA(02D5h)   - GPUREG_VSH_OPDESCS_INDEX (W)
10401B58h+i*4   - PICA(02D6h+i) - GPUREG_VSH_OPDESCS_DATA(0..7) (W)
```

3DS GPU Internal Registers - Shader 2 (Unknown purpose)

```
10401B80h       - PICA(02E0h)   - GPUREG_VSH2_BOOLUNIFORM (R/W)
10401B84h+i*4   - PICA(02E1h+i) - GPUREG_VSH2_INTUNIFORM_I0..I3 (R/W)
10401BA4h       - PICA(02E9h)   - GPUREG_VSH2_INPUTBUFFER_CONFIG (R/W)
10401BA8h       - PICA(02EAh)   - GPUREG_VSH2_ENTRYPOINT (R/W)
10401BACh       - PICA(02EBh)   - GPUREG_VSH2_ATTR_PERMUTATION_LOW (R/W)
10401BB0h       - PICA(02ECh)   - GPUREG_VSH2_ATTR_PERMUTATION_HIGH (R/W)
10401BB4h       - PICA(02EDh)   - GPUREG_VSH2_OUTMAP_MASK (R/W)
10401BBCh       - PICA(02EFh)   - GPUREG_VSH2_CODETRANSFER_END (W)
10401BC0h       - PICA(02F0h)   - GPUREG_VSH2_FLOATUNIFORM_INDEX (W)
10401BC4h+i*4   - PICA(02F1h+i) - GPUREG_VSH2_FLOATUNIFORM_DATA(0..7) (W)
10401BECh       - PICA(02FBh)   - GPUREG_VSH2_CODETRANSFER_INDEX (W)
10401BF0h+i*4   - PICA(02FCh+i) - GPUREG_VSH2_CODETRANSFER_DATA(0..7) (W)
10401C14h       - PICA(0305h)   - GPUREG_VSH2_OPDESCS_INDEX (W)
10401C18h+i*4   - PICA(0306h+i) - GPUREG_VSH2_OPDESCS_DATA(0..7) (W)
```

3DS GPU Internal Registers - Shader 3 (Unknown purpose)

```
10401C40h       - PICA(0310h)   - GPUREG_VSH3_BOOLUNIFORM (R/W)
10401C44h+i*4   - PICA(0311h+i) - GPUREG_VSH3_INTUNIFORM_I0..I3 (R/W)
10401C64h       - PICA(0319h)   - GPUREG_VSH3_INPUTBUFFER_CONFIG (R/W)
10401C68h       - PICA(031Ah)   - GPUREG_VSH3_ENTRYPOINT (R/W)
10401C6Ch       - PICA(031Bh)   - GPUREG_VSH3_ATTR_PERMUTATION_LOW (R/W)
10401C70h       - PICA(031Ch)   - GPUREG_VSH3_ATTR_PERMUTATION_HIGH (R/W)
10401C74h       - PICA(031Dh)   - GPUREG_VSH3_OUTMAP_MASK (R/W)
10401C7Ch       - PICA(031Fh)   - GPUREG_VSH3_CODETRANSFER_END (W)
10401C80h       - PICA(0320h)   - GPUREG_VSH3_FLOATUNIFORM_INDEX (W)
10401C84h+i*4   - PICA(0321h+i) - GPUREG_VSH3_FLOATUNIFORM_DATA(0..7) (W)
10401CACh       - PICA(032Bh)   - GPUREG_VSH3_CODETRANSFER_INDEX (W)
10401CB0h+i*4   - PICA(032Ch+i) - GPUREG_VSH3_CODETRANSFER_DATA(0..7) (W)
10401CD4h       - PICA(0335h)   - GPUREG_VSH3_OPDESCS_INDEX (W)
10401CD8h+i*4   - PICA(0336h+i) - GPUREG_VSH3_OPDESCS_DATA(0..7) (W)
```

3DS GPU External Registers - Memory Control/Status Registers

The 3DS bootrom error screen initializes only 10400004h and 10400030h (VRAM control/power). The 3DS OS does reportedly also initialize 10400050h and 10400054h (timings?).

10400000h - GPU_FIXED_00010002h (always 00010002h, maybe ChipID/Version) (R)

```
0-31  Always 00010002h (maybe ChipID/Version)
```

Note: Register 104010D4h seems to contain the same readonly value.

10400004h - GPU_VRAM_CNT (reset=0, bootrom=300h) (R/W)

```
  0-31  Unknown (0..FFFFFFFFh)
```

Seems to have no effect on LCD output. However, value FFFFFFFFh does mess up the polygon renderer (or MEMCOPY unit), causing each 4 scanlines to be drawn twice.

10400008h - GPU_UNKNOWN_2BIT (reset=0) (R/W)

```
  0-1   Unknown (0..03h)
  2-31  Unused (0)
```

10400030h - GPU_VRAM_POWER (R/W)

```
  0-7   Unknown (0..FFh)
  8     Power off VRAM_A data.bit0-63   (0=On, 1=Power Off)
  9     Power off VRAM_A data.bit64-127 (0=On, 1=Power Off)
  10    Power off VRAM_B data.bit0-63   (0=On, 1=Power Off) ;\later uses VRAM_A
  11    Power off VRAM_B data.bit64-127 (0=On, 1=Power Off) ;/when VRAM_B=off?
  12-31 Unknown (0..FFFFFh)
```

Seems to disable VRAM power, VRAM_A/B are 3Mbyte each, and each of them seems to have separate sections for data bit0-63 and 64-127 (ie. the first and second 8 bytes within 16 byte snippets).
After writing to 10400030h, one should issue a dummy read from 10400030h.
Even when disabled only for a short moment, the VRAM does immediately forget it's data and gets filled with random values; this can be seen after re-enabling the memory (the CPU hangs when reading VRAM while it is disabled).

10400034h - GPU_STAT_IRQ_FLAGS (R)

Contains IRQ flags (normally IRQ handling/polling should work fine without needing this register; except, polling GPU_MEMCOPY_CNT can apparently hang the CPU, whilst polling GPU_STAT_IRQ_FLAGS.bit30 does work better).

```
  0     Unknown, usually set ;\get cleared when setting GPU_MAKE_SOMETHING.bit0
  1     Unknown, usually set ;/(unknown if/when the flags can get set again)
  2-25  Unused (0)
  26    IRQ 28h PSC0 aka GPU_MEMFILL 0 ready ;mirror of GPU_MEMFILL_CNT0.bit1
  27    IRQ 29h PSC1 aka GPU_MEMFILL 1 ready ;mirror of GPU_MEMFILL_CNT1.bit1
  28    IRQ 2Ah PDC0 aka GPU H/V-IRQ for top screen
  29    IRQ 2Bh PDC1 aka GPU H/V-IRQ for bottom screen
  30    IRQ 2Ch PPF  aka GPU_MEMCOPY ready   ;mirror of GPU_MEMCOPY_CNT.bit8
  31    IRQ 2Dh P3D  aka GPUREG_IRQ_CMP/REQ match (commonly at end of cmdlist)
```

Bit26,27,30 can be cleared in the corresponding mirrored registers.
Bit28,29 are automatically set/cleared during LCD output (usually once per frame).
Bit31 can be cleared by setting IRQ_REQ and IRQ_CMP to different values.

10400038h - GPU_STAT_SOMETHING (R)

```
  0-31  Can be 00000000h, 10400800h, 10402000h, or 10410000h
```

Unknown, initially 10402000h on reset, 10400800h after LCD init or so, and becomes 00000000h after setting GPU_MAKE_SOMETHING.bit0, and (thereafter?) changes to 10410000h. No idea if 104xxxxxh is related to the address of the GPU I/O area.

1040003Ch - GPU_MAKE_SOMETHING (R/W)

```
  0     Unknown (0=Normal, 1=GPU_STAT_SOMETHING changes after a while)
  1-31  Unknown (0..7FFFFFFFh)
```

Unknown, setting bit0 (even only for a short moment) will later on cause GPU_STAT_SOMETHING to become 10410000h. Also affects GPU_STAT_IRQ_FLAGS.bit0-1.

10400040h - GPU_STAT_BACKLIGHT_OR_SO (R)

```
  0-1   Unknown (usually 0, but can be 3 when [104000C0h]=0 or [104000CCh]=0)
  2-31  Unused (0)
```

10400044h - GPU_UNKNOWN_CAN_BE_7F80h (R)

```
  0-31  Texture related...? (usually 0, but sometimes 7F80h, and once 1200h)
```

Unknown. Usually zero, but can contain other values after rendering with GPUREG_TEXUNIT_CONFIG=nonzero (even then, it does often stay zero, but sometimes changes to 7F80h, and once changed to 1200h). Whether the register does change seems to depend on power-up state of other uninitialized registers/memory and/or whatever timings (if nonzero values did occur then they do usually reappear upon next rendering even after reset, but that behaviour can change on next power-up).
Note: Value 1200h occurred after changing GPUREG_COLORBUFFER_LOC and GPUREG_DEPTHBUFFER_LOC to dummy address zero (but that couldn't be reproduced).

10400048h - GPU_UNKNOWN_32BIT (reset=0) (R/W)

```
  0-31  Unknown (0..FFFFFFFFh)
```

10400050h - GPU_TIMING_CNT1 (R/W=FFFFFFFFh) (res=1111EF00h) ;init=22221200h

```
  0-31  Unknown (0..FFFFFFFFh) (affects GPU_MEMCOPY timings and maybe others)
```

Somehow affects timings. Different values can make timings about 10% faster or slower (or, some values seem to make no difference at all). For example,

```
  10101010h or 20202020h --> slower MEMCOPY
  1111EF00h or 22221200h --> normal MEMCOPY
  FFFFFFFFh              --> faster MEMCOPY
```

10400054h - GPU_TIMING_CNT2 (R/W=FFFFFFFFh) (res=00000112h) ;init=00000FF2h

```
  0-31  Unknown (0..FFFFFFFFh) (probably affects whatever timings)
```

10400058h - GPU_STAT_BUSY_FLAGS (R)

```
  10    Unknown, seems to be set on power-up, but later cleared?
  17    Often set after writing GPU_VRAM_POWER
  18    Sometimes set after writing GPU_VRAM_POWER
  19    GPU_MEMFILL_0/1 Busy (0=No, 1=Busy)
  20    GPU_MEMCOPY Busy     (0=No, 1=Busy; gets set some cycles after start)
```

1040005Ch - GPU_MAKE_WHATEVER_MESS (R/W)

```
  0     Whatever, gets copied to bit8                   (W)
  8     Whatever, contains value written to bit0        (R)
  16-21 Whatever, 00h..3Fh                              (R/W)
  24-29 Whatever, 00h..3Fh                              (R/W)
```

Affects the 30h-word pattern returned when reading below register.

10400060h - GPU_STAT_WHATEVER_MESS (R)

```
  0-6   Unused? (0)
  7-22  Whatever Mess
  23    Unused? (0)
  24-29 Whatever Mess
  30-31 Unused? (0)
```

Repeatly reading returns a mess, repeated every 30h words, for example:

```
  08002400h, 01404180h, 04002000h, 00020000h, 20000200h, 02080000h, 00080000h,
  0041A000h, 11006100h, 00041000h, 00242000h, 00100880h, 1200A000h, 08000000h,
  10042080h, 04082400h, 00002400h, 00000000h, 1040C500h, 03000000h, ...
```

The same values as above are also returned when inserting delays in the read loop. So it's either random generated... or a FIFO that reads junk from an uninitialized array?
Changing 1040005Ch (eg. writing FFFFFFFFh) causes above to return different garbage values (also reapeating each 30h words, but containing a more regular pattern with fewer bits set).

10400068h GPU_UNKNOWN_INITIALLY_00A80000h (res=00A80000h) (R/W)

```
  0-31  Unknown (0..FFFFFFFFh) (initially 00A80000h on reset)
```

10400070h - Read-only memory traffic counters (R)

```
  10400070h  GPU_STAT_TOTAL_NON_VRAM_READS      ;\Non-VRAM (eg. AXI,QTM,FCRAM)
  10400074h  GPU_STAT_TOTAL_NON_VRAM_WRITES     ;/
  10400078h  GPU_STAT_TOTAL_VRAM_A_READS        ;\1st 3Mbyte VRAM block
  1040007Ch  GPU_STAT_TOTAL_VRAM_A_WRITES       ;/
  10400080h  GPU_STAT_TOTAL_VRAM_B_READS        ;\2nd 3Mbyte VRAM block
  10400084h  GPU_STAT_TOTAL_VRAM_B_WRITES       ;/
  10400088h  GPU_STAT_POLYGON_ARRAY_READS       ;-ATTR_BUF, INDEX_LIST
  1040008Ch  GPU_STAT_POLYGON_TEXTURE_READS     ;-cache misses only
  10400090h  GPU_STAT_POLYGON_DEPTHBUFFER_READS
  10400094h  GPU_STAT_POLYGON_DEPTHBUFFER_WRITES
  10400098h  GPU_STAT_POLYGON_COLORBUFFER_READS
  1040009Ch  GPU_STAT_POLYGON_COLORBUFFER_WRITES
  104000A0h  GPU_STAT_LCD_UPPER_SCREEN_READS    ;\for both left+right buffer
  104000A4h  GPU_STAT_LCD_LOWER_SCREEN_READS    ;/
  104000A8h  GPU_STAT_MEMCOPY_SRC_READS         ;\GPU_MEMCOPY
  104000ACh  GPU_STAT_MEMCOPY_DST_WRITES        ;/
  104000B0h  GPU_STAT_MEMFILL_0_DST_WRITES      ;\GPU_MEMFILL 0/1
  104000B4h  GPU_STAT_MEMFILL_1_DST_WRITES      ;/
  104000B8h  GPU_STAT_CPU_READS_FROM_VRAM_A_B   ;\counted by GPU because the
  104000BCh  GPU_STAT_CPU_WRITES_TO_VRAM_A_B    ;/GPU must forward vram to cpu
```

104000C4h - GPU_BASE_ADDR_VRAM_A (reset=18000000h) (R/W)
104000C8h - GPU_BASE_ADDR_VRAM_B (reset=18300000h) (R/W)

```
  0-31  Base address for VRAM_A/B blocks (0..FFFFFFFFh) (any alignment needed?)
```

The 3DS contains two 3MByte VRAM blocks (ie. 6Mbyte in total). These registers allow to swap the base address of those two blocks, or to map them to other locations within the following 12Mbyte region:

```
  18000000h..182FFFFFh  Usually VRAM_A (3Mbyte)
  18300000h..185FFFFFh  Usually VRAM_B (3Mbyte)
  18600000h..18BFFFFFh  Usually empty/zerofilled (6Mbyte)
```

Trying to map VRAM outside of that 12Mbyte region might work, but the ARM CPU will trigger data aborts when trying to access that memory.

104000C0h - GPU_BACKLIGHT_OR_SO_1 (reset=20000000h) (R/W)
104000CCh - GPU_BACKLIGHT_OR_SO_2 (reset=20000000h) (R/W)

```
  0-31  Unknown (0..FFFFFFFFh)
```

Reportedly 104000C0h does "Writes 0 to allow backlights to turn off, 20000000h to force them always on." Unknown what that means. Default is 20000000h (and backlights CAN be switched off in that state). Maybe "allow" means that they can switch off automatically... upon GPU inactivity... or upon blank screen... or upon screen saver timeouts... or whatever? Or maybe 20000000h is FCRAM base, completely unrelated to backlights? Note: Setting [104000C0h] or [104000CCh] to zero can cause [10400040h] to get nonzero.

104000D0h - GPU_UNKNOWN_4BIT (reset=0) (R/W)

```
  0-3   Unknown (0..0Fh)
  4-31  Unused (0)
```

3DS GPU External Registers - Top/Bottom Screen and Framebuffer Setup

104004xxh = Framebuffer Setup "PDC0" (top screen)
104005xxh = Framebuffer Setup "PDC1" (bottom screen)
PDC = "PICA Display Control" or so?

Top/Bottom Screen Setup Register Summary

```
  10400400h/10400500h 4   V-Total-1                ;1C2h       ;\
  10400404h/10400504h 4   V-Lower-border-end       ;0D1h       ;
  10400408h/10400508h 4   V-Upper-border-middle    ;1C1h       ;
  1040040Ch/1040050Ch 4   V-Upper-border-end       ;1C1h       ; V
```

```
10400410h/10400510h 4   V-Sync-start              ;000h/0CDh  ;
10400414h/10400514h 4   V-Sync-end                ;0CFh       ;
10400418h/10400518h 4   V-Lower-border-start      ;0D1h       ;
1040041Ch/1040051Ch 2   V-IRQ-start               ;1C1h       ;
1040041Eh/1040051Eh 2   V-IRQ-end                 ;1C5h       ;/
10400420h/10400520h 2   V-Pre-padding fetch start?;000h       ;\V
10400422h/10400522h 2   V-??   ;must be <=1C2h    ;001h       ;/
10400424h/10400524h 4   H-Total-1                 ;19Dh       ;\
10400428h/10400528h 4   H-Left-border-end         ;002h/052h  ;
1040042Ch/1040052Ch 4   H-Right-border-middle     ;192h       ;
10400430h/10400530h 4   H-Right-border-end        ;192h       ; H
10400434h/10400534h 4   H-Sync-Start             ;192h/04Fh  ;
10400438h/10400538h 4   H-Sync-End               ;001h/050h  ;
1040043Ch/1040053Ch 4   H-Left-border-start       ;002h/052h  ;
10400440h/10400540h 2   H-IRQ-start (or end)      ;192h       ;
10400442h/10400542h 2   H-IRQ-end (or start)      ;193h       ;/
10400444h/10400544h 4   V-Increment-H            ;000h       ;-V
10400448h/10400548h 4   Screen Vsync/Hsync type
1040044Ch/1040054Ch 4   Screen Border color
10400450h/10400550h 4   V-Current position (R)
10400454h/10400554h 4   H-Current position (R)
10400458h/10400558h 04h Unused (0)
1040045Ch/1040055Ch 2   V-Picture size            ;0F0h       ;-V
1040045Eh/1040055Eh 2   H-Picture size (no effect);190h/140h  ;-H
10400460h/10400560h 2   V-Lower-border-middle     ;0D1h       ;\V
10400462h/10400562h 2   V-Upper-border-start      ;1C1h       ;/
10400464h/10400564h 2   H-Left-border-middle      ;002h/052h  ;\H
10400466h/10400566h 2   H-Right-border-start      ;192h       ;/
10400468h/10400568h 4   Framebuffer 0 address for Left eye (or both eyes)
1040046Ch/1040056Ch 4   Framebuffer 1 address for Left eye (or both eyes)
10400470h/10400570h 4   Framebuffer format
10400474h/10400574h 4   Interrupt type
10400478h/10400578h 4   Framebuffer Select and Request/Ack
1040047Ch/1040057Ch 4   Screen Status flags (R)
10400480h/10400580h 4   Color Lookup Table index (R/W)
10400484h/10400584h 4   Color Lookup Table data (R/W)
10400488h/10400588h 08h Unused (0)
10400490h/10400590h 4   Framebuffer Horizontal Address Step
10400494h/10400594h 4   Framebuffer 0 address for Right eye (or unused
10400498h/10400598h 4   Framebuffer 1 address for Right eye (or unused
1040049Ch/1040059Ch 2   V-Latching-Point          ;000h       ;-V
1040049Eh/1040059Eh 2   H-Latching-Point          ;192h       ;-H
104004A0h/104005A0h 60h Unused (0)
```

Most of the registers do have internal copies (the H/V registers, and Border color, and maybe some others), changing these registers doesn't have any effect on the visible picture & timings - until reaching the H/V-Latching Point (which does forward the registers to their internal copies).

Control/Status Registers

10400474h/10400574h - Interrupt Type (R/W)

```
0      Display Enable (should be 1) (0=Off/screen fades out, 1=On/normal)
1-7    Unused (0)
8-10   Interrupt Mode (5=OncePerFrame, 7=None, 0..4,6=Special) (see below)
7-11   Unused (0)
16     Unknown...?    (0=???, 1=Normal)
17-31  Unused (0)
```

The default is Interrupt Mode 5 (which acts as normal framerate interrupt).
Interrupt Mode 0,1,2,3 do require manually acknowledging/requesting the next IRQ via Request register bit18 (and bit17 in some cases).

```
Interrupt Mode 0:
- Triggers here or there depending on H/V settings
Interrupt Mode 1:
- Triggers at V=[10400544h], H=[10400540h]              ;irq point
```

```
- Triggers at V=[10400504h], H=[10400528h]..[10400566h]  ;picture area
Interrupt Mode 2:
- Triggers here or there depending on H/V settings
Interrupt Mode 3:
- Triggers at V=[10400504h], H=[10400528h]..[10400566h]  ;picture area
Interrupt Mode 4 (in selected H's)
- Triggers at V=[1040051Ch], and H=[10400542h]..[10400540h]
Interrupt Mode 5 (framerate interrupt, usually at H=Hblank)
- Triggers at V=[10400544h] and H=[10400540h]
Interrupt Mode 6 (once on every H)
- Triggers at V=[1040051Ch], and H=any
Interrupt Mode 7 (interrupts disabled)
- Doesn't trigger any IRQs.
```

For all IRQ H/V timings, mind that the display is drawn from left to right (unlike normal top-to-bottom displays).
Note: Most of the H positions do actually trigger at H+1 (and perhaps similar for V positions, though those are more difficult to measure).

10400478h/10400578h - Framebuffer Select and Request/Ack (R/W)

```
0      Display Framebuffer (0=Buffer 0, 1=Buffer 1)
1-3    Unused (0)
4      Request STAT.bit4 to get triggered (0=No, 1=Request/Busy)
5-7    Unused (0)
8      Unknown...?   (0=Normal, 1=???)
9-15   Unused (0)
16     Unknown...?   (0=Normal, 1=???)
17     Unknown...?   (0=Normal, 1=???, set in bootrom IRQ handler, but why?)
18     Acknowledge STAT.bit15? resurrect after ONESHOT irq (unless,
         bit18/ack works ONLY if OUTSIDE of display area?)
19-31  Unused (0)
```

Apart from bit0, all bits in the register are kind of "Request/Busy" flags (and get cleared once when the request is handled).

```
bit4:          cleared at H=[10400540h]..[10400542h]
bit8,16,17,18: cleared after a handful of clock cycles
```

Request.bit4 does soemhow relate to the framebuffer selection in bit0 (see STAT.bit4 for details)... Guess: maybe bit4 prevents bit0 from being internally applied until hblank?

1040047Ch/1040057Ch - Screen Status flags (R)

```
0      H-IRQ-area, when H=[10400540h]..[10400542h]
1      V-IRQ-area, when H=[1040051Ch]..[1040051Eh]
2-3    Unused (0)
4      Can get set if [10400578h].bit4=1, and depending on [10400578h].bit0:
         if [10400578h].bit4=0: at H=[10400540h]..[10400528h]  ;ONCE
         if [10400578h].bit4=1: at H=[10400528h]..FOREVER      ;FOREVER
         that, happening at V=[10400544h] (in both of the above two cases)
         note: ONCE means that the bit goes ON-and-OFF once (and stays off)
         note: FOREVER means that the bit goes ON (and stays on)
5-7    Unused (0)
8      H-Sync,    H=[10400534h]..[10400538h]
9      H-Blank,   H=[10400530h]..[1040053Ch] (between borders)
10     H-Picture, H=[10400528h]..[10400566h] (between borders)
11     Unused (0)
12     V-Sync,    H=[10400510h]..[10400514h]
13     V-Blank,   H=[1040050Ch]..[10400518h] (between borders)
14     V-Picture, H=[10400504h]..[10400562h] (between borders)
15     Normally 0, but can be set, if interrupts enabled IRQ request?
         bit15: set at H/V-picture start, sticky bit?
         but, WHY is bit15 is never set with bootrom default settings?
         related to 1040059Ch,1040059Eh !!!
16-31  Unused (0)
```

10400448h/10400548h - Screen Vsync/Hsync type (R/W)

```
  0      Seems to disable HSync (or Vsync?), or sync mode/polarity or so?
  1-7    Unused (0)
  8      Seems to disable VSync (or Hsync?), or sync mode/polarity or so?
  9-31   Unused (0)
```
Should be 00h/11h for top/bottom screen. Maybe selects using /SYNC signals or types like "DE only" with sync on Display Enable? The pinout for bottom screen is said to include /HSYNC and /VSYNC pins. The pinout for top screen is unknown.

_____ Framebuffer Registers _____

10400468h/10400568h - Framebuffer 0 address for Left eye (or both eyes)
1040046Ch/1040056Ch - Framebuffer 1 address for Left eye (or both eyes)
10400494h/10400594h - Framebuffer 0 address for Right eye (or unused)
10400498h/10400598h - Framebuffer 1 address for Right eye (or unused)
```
  0-3    Unused (0)
  4-31   Address, in 16-byte units
```

10400490h/10400590h - Framebuffer Horizontal Address Step (R/W)
```
  0-3    Unused (0)
  4-31   Address step, in 16-byte units (usually 3Ch, aka 240pix*4byte/16)
```

10400470h/10400570h - Framebuffer format (R/W)
```
  0-2    Framebuffer RGBA Format (0=8888, 1=8880, 2=5650, 3=5551, 4..?=4444)
           Note: The alpha in RGBA types is just skipped/ignored.
  3      Unused (0)
  4-5    Horizontal Zoom (0=Normal/HiRes, 1=Double/LowRes, 2/3=LeftRightBuffer)
  6      Vertical Zoom   (0=Normal/HiRes, 1=Double/LowRes)
  7      Unused (0)
  8-9    Unknown (0..3) (should be 0)
           00h = Normal (should be normally used)
           01h = Unknown "get rid of rainbow strip on top of screen" uh? mess!
           02h = Unknown                (actually distorted display?)
           03h = Unknown "black screen" (actually normal display?)
  10-15  Unused (0)
  16-31  Unknown (0..FFFFh) (should be 8)
            (FFFFh=slow horizontal drawing?)
            maybe... dotclk divider (and perhaps multiplier in other bits)
            hmmm, but, very large values cause vram-misaddressing?
```
Obscure Color Format Notes:
Color components are laid out in reverse byte order, with the most significant bits used first (ie. non-24bit pixels are stored as a little-endian values). For instance, a raw data stream of two GL_RGB565_OES pixels looks like GGGBBBBB RRRRRGGG GGGBBBBB RRRRRGGG. uh, that are TWO pixels, but which one is shown first/bottom? uh, and that's for non-24bit, so, how does 24bit look like?

_____ Color Registers _____

10400480h/10400580h - Color Lookup Table index (R/W)
```
  0-7    Index        ;reportedly write-only (uh, but actually, it is R/W)
  8-31   Unused (0)
```

10400484h/10400584h - Color Lookup Table data (R/W)
```
  0-7    Red[index]  ;\
  8-15   Green[index]; is that always RGB, or could it be swapped to BGR or so?
  16-23  Blue[index] ;/
  24-31  Unused (0)
```
Contains the value of the color lookup table indexed by the above register. Accessing this register will increase the index register by one.
For RGBA 8888/8880, the table should be usually filled with linear increasing values: 000000h, 010101h, 020202h, 030303h, .., FFFFFFh. Or alternately with non-linear values if the displays require 'gamma'

adjustments.
Unknown which indices are used in RGBA 5650, 5551, and 4444 modes.

1040044Ch/1040054Ch - Screen Border color (R/W)

```
  0-7    Red    (00h..FFh)
  8-15   Green  (00h..FFh)
  16-23  Blue   (00h..FFh)
  24-31  Unused (0)
```

The screen border isn't used for fullscreen pictures, but it could be used if the picture is smaller than the actual screen. To disable the border, set border start/end registers to the same value.
There are also "border middle" registers (with unknown purpose and without visible effect, except, the middle values for vertical border MUST be within border start/end range, otherwise the screen dies with fading out).

```
 _____________________________ H/V-Registers _____________________________
```

Vertical Position Registers

```
  10400444h/10400544h 4   V-Increment-H             ;000h
  1040049Ch/1040059Ch 2   V-Latching-Point          ;000h
  10400420h/10400520h 2   V-Pre-padding fetch start?;000h
  10400422h/10400522h 2   V-??   ;must be <=1C2h    ;001h
  10400410h/10400510h 4   V-Sync-start              ;000h/0CDh
  10400414h/10400514h 4   V-Sync-end                ;0CFh
  10400418h/10400518h 4   V-Lower-border-start      ;0D1h
  10400460h/10400560h 2   V-Lower-border-middle     ;0D1h
  10400404h/10400504h 4   V-Lower-border-end        ;0D1h
  1040045Ch/1040055Ch 2   V-Picture size            ;0F0h
  10400462h/10400562h 2   V-Upper-border-start      ;1C1h
  10400408h/10400508h 4   V-Upper-border-middle     ;1C1h
  1040040Ch/1040050Ch 4   V-Upper-border-end        ;1C1h
  1040041Ch/1040051Ch 2   V-IRQ-start               ;1C1h
  1040041Eh/1040051Eh 2   V-IRQ-end                 ;1C5h
  10400400h/10400500h 4   V-Total-1                 ;1C2h
  10400450h/10400550h 4   V-Current position (R)    ;000h..1C2h
```

These are usually using same values for top/bottom screen (with 240pix height each), except that Nintendo has used different Sync length for some reason.

Horizontal Position Registers

```
  10400434h/10400534h 4   H-Sync-Start              ;192h/04Fh
  10400438h/10400538h 4   H-Sync-End                ;001h/050h
  1040043Ch/1040053Ch 4   H-Left-border-start       ;002h/052h
  10400464h/10400564h 2   H-Left-border-middle      ;002h/052h
  10400428h/10400528h 4   H-Left-border-end         ;002h/052h
  1040045Eh/1040055Eh 2   H-Picture size (no effect);190h/140h
  10400466h/10400566h 2   H-Right-border-start      ;192h
  1040042Ch/1040052Ch 4   H-Right-border-middle     ;192h
  10400430h/10400530h 4   H-Right-border-end        ;192h
  1040049Eh/1040059Eh 2   H-Latching-Point          ;192h
  10400440h/10400540h 2   H-IRQ-start (or end)      ;192h
  10400442h/10400542h 2   H-IRQ-end (or start)      ;193h
  10400424h/10400524h 4   H-Total-1                 ;19Dh
  10400454h/10400554h 4   H-Current position (R)    ;000h..19Dh
```

These are using the same Htotal for top/bottom screen, but different picture/blanking widths (for top/bottom screens with 400pix/320pix width). The top screen does also support a 800pix hires mode (which should use double Htotal, and adjust other registers accordingly).

Note: The values in the H/V registers are 12bit wide, 0..FFFh (the upper 4bit/20bit of the 16bit/32bit registers are unused, always zero).

Wrong H/V values may cause various effects - which may differ depending on the display type. New3DS bottom screen tends to output a stable image (with fading upper/right areas if it is too small), or no picture at all (with

whole screen fading). However, other screens are said to lose sync or misalign lines and such stuff.

Setting the V-pre-padding bigger than V-lower-border end causes the bottom-most picture lines to display the most recent pixel color instead of fetching new pixels from VRAM. A Similar effect occurs in upper picture area when setting V-picture-size too small.

Misc Notes

Framebuffers

These LCD framebuffers normally contain the last rendered frames from the GPU. The framebuffers are drawn from left-to-right, instead of top-to-bottom. Thus the beginning of the framebuffer is drawn starting at the left side of the screen.

Both of the 3D screen left/right framebuffers are displayed regardless of the 3D slider's state, however when the 3D slider is set to "off" the 3D effect is disabled. Normally when the 3D slider's state is set to "off" the left/right framebuffer addresses are set to the same physical address. When the 3D effect is disabled and the left/right framebuffers are set to separate addresses, the LCD seems to alternate between displaying the left/right framebuffer each frame.
Uh, it can alternate left/right buffers each frame? I have NEVER seen that effect when testing register settings (and it would have caused massive flicker because I didn't had initialized the right buffer). What it can do is alternate left/right buffer after each pixel horizontally (including on bottom screen).

Init Values from nngxInitialize for Top Screen

Unknown if below values are "better" than bootrom, or if they are typos.

```
10400400h = 1C2h
10400404h = D1h
10400408h = 1C1h
1040040Ch = 1C1h
10400410h = 0
10400414h = CFh
10400418h = D1h
1040041Ch = 1C501C1h
10400420h = 10000h
10400424h = 19Dh
10400428h = 2
1040042Ch = 1C2h ??? ;bootrom: 192h ?
10400430h = 1C2h ??? ;bootrom: 192h ?
10400434h = 1C2h ??? ;bootrom: 192h ?
10400438h = 1
1040043Ch = 2
10400440h = 1960192h ;bootrom: 1930192h ?
10400444h = 0
10400448h = 0
1040045Ch = 19000F0h   ;msbs=width=400, lsbs=height=240
10400460h = 1C100D1h
10400464h = 1920002h
10400470h = 80340h   ;bootrom: 80040h ?
1040049Ch = 0        ;bootrom: 1920000h ?
```

More Init Values from nngxInitialize for Top Screen

```
10400468h = 18300000h  ;\later changed by GSP module when updating state,
1040046Ch = 18300000h  ;/framebuffer
10400494h = 18300000h
10400498h = 18300000h
10400478h = 1          ;-doesn't stay 1, read as 0
10400474h = 10501h
```

Reportedly 10400478h/10400578h.bit0 "doesn't stay 1, read as 0", uh, actually that applies for ALL OTHER bits in that register, but NOT for bit0.

Further Registers
Apart from above GPU registers, the screens do also need initialization of the "LCD" registers,
3DS Video LCD Registers
plus, initialization of whatever I2C registers for each screen. Plus some MCU/I2C registers for backlight or so.

3DS GPU External Registers - Memfill and Memcopy

```
_____________________________ Memory Fill _____________________________
```

10400010h/10400020h - GPU_MEMFILL_DST_ADDR0/1 - Memfill 0/1 Start Address
10400014h/10400024h - GPU_MEMFILL_DST_END0/1 - Memfill 0/1 End Address

```
0      Unused (0)
1-28   Physical Memory Address, in 16-byte units
29-31  Unused (0)
```

10400018h/10400028h - GPU_MEMFILL_DATA0/1 - Memfill 0/1 Fill Value

```
0-31   Fill Value
```

1040001Ch/1040002Ch - GPU_MEMFILL_CNT0/1 - Memfill 0/1 Control

```
0      Start/Busy   (0=Idle/Ready, 1=Start/Busy)                      (R/W)
1      Ready IRQ    (0=No, 1=Ready, IRQ 28h/29h) (write 0 to clear)   (R/ack)
2-7    Unused (0)
8-9    Fill-width   (0=16bit, 1=24bit, 2=32bit, 3=Same as 1)          (R/W)
10-15  Unused (0)
16-20  Unknown, 5bit, R/W                                             (R/W)
21-31  Unused (0)
```

A memory fill is triggered by setting bit0 in the control register. Doing so aborts any running memory fills on that filling unit. Upon completion, the hardware clears bit0 and sets bit1 and fires interrupt PSC0/PSC1 (IRQ 28h/29h).

```
_____________________________ Memory Copy _____________________________
```

10400C00h - GPU_MEMCOPY_SRC_ADDR - Memcopy Input physical address (R/W)
10400C04h - GPU_MEMCOPY_DST_ADDR - Memcopy Output physical address (R/W)

```
0      Unused (0)
1-28   Physical Memory Address, in 16-byte units
29-31  Unused (0)
```

10400C08h - GPU_MEMCOPY_DISPLAY_SIZE - DisplayTransfer width/height(R/W)

```
0-2    Unused (0)
3-15   Input Height (in 8-pixel units, usually 240/8 for 3DS)
16-18  Unused (0)
19-31  Input Width  (in 8-pixel units, usually 320/8 or 400/8 for 3DS)
```

Input size before downscale (the output size is the same, unless downscale was used).

10400C0Ch - GPU_MEMCOPY_DISPLAY_GAP - DisplayTransfer Input height+gap (R/W)

```
0-2    Unused (0)
3-15   Input Height+Gap (in 8-pixel units, usually (240+0)/8 for 3DS)
16-31  Unused (0)
```

Allows to skip input pixels, used only if GPU_MEMCOPY_FLAGS.bit2=1.

10400C10h - GPU_MEMCOPY_FLAGS - Memcopy Transfer flags (R/W)

```
0      Horizontal Flip    (0=Normal, 1=Mirror)
1      Conversion Mode    (0=TiledToLinear, 1=LinearToTiled)
2      Input Gap          (0=Use raw height, 1=Use height+gap)
3      Copy Mode          (0=DisplayCopy/Pixels, 1=TextureCopy/RawBytes)
4      Unused (0)
```

```
  5      Conversion Disable (0=Convert as specified in bit1, 1=TiledToTiled)
  6-7    Unused (0)
  8-10   Input RGBA Format  (0=8888, 1=8880, 2=5650, 3=5551, 4..7=4444)
  11     Unused (0)
  12-14  Output RGBA Format (0=8888, 1=8880, 2=5650, 3=5551, 4..7=4444)
  15     Unused (0)
  16     Block Tiling Size  (0=Normal/8x8 pixels, 1=32x32 pixels)
  17-23  Unused (0)
  24-25  Output Downscale   (0=No, 1=Height/2, 2=WidthAndHeight/2, 3=Invalid)
  26-31  Unused (0)
```
The transfer hangs on some Input/Output RGBA combinations:
```
  32bit Input (8888) works with all output formats
  24bit Input (8880) works only if output is also 24bit
  16bit Input (5650/5551/4444) works only if output is also 16bit
```

10400C14h - GPU_MEMCOPY_UNKNOWN_21BIT - Memcopy (R/W)
```
  0-20   Unknown, R/W
  21-31  Unused (0)
```
"GSP module writes value 0 here prior to writing to 10400C18h, for cmd3."

10400C18h - GPU_MEMCOPY_CNT - Memcopy Start/Busy (R/W)
```
  0      Start/Busy (0=Idle/Ready, 1=Start/Busy)                        (R/W)
  1-7    Unused (0)
  8      Ready IRQ  (0=No, 1=Ready, "PPF" IRQ 2Ch) (write 0 to clear)   (R/ack)
  9-31   Unused (0)
```
Caution: Polling this register can HANG the ARM11 CPU at end of transfer; when using polling, it's better to wait for GPU_STAT_IRQ_FLAGS.bit30=1.

10400C1Ch - GPU_MEMCOPY_REMAIN_IRQ - Memcopy Remain IRQ (R/W)
```
  0-13   Transfer IRQ position (compare value for below remain counter)   (R/W)
  14-15  Unused (0)
  16-29  Transfer width remain, in 4-pixel units (3FFEh..0, or 3FFFh=done)  (R)
  30-31  Unused (0)
```
The IRQ position should be set to 3FFFh for triggering IRQ flag at transfer end, otherwise the IRQ would trigger before completion (or never at all).
Example: For width=320pix, remain goes through 4Fh..00h, and does then wrap to 3FFFh.
The 4-pixel unit might be due to decreasing the counter upon both reading & writing 8-pixel chunks.
Used for DisplayTransfer. And maybe also for TextureCopy?

10400C20h - GPU_MEMCOPY_TEX_SIZE - TextureCopy total amount
```
  0-3    Unused (0)
  4-31   Total amount of data to copy, in 16-byte units
```

10400C24h - GPU_MEMCOPY_TEX_SRC_WIDTH - TextureCopy input line width/gap
10400C28h - GPU_MEMCOPY_TEX_DST_WIDTH - TextureCopy output line width/gap
```
 For SRC_WIDTH:
  0-15   Width, in unknown units (maybe 16-byte units, same as below?)
  16-31  Gap, in 16-byte units
 For DST_WIDTH:
  0-31   Width and Gap (probably same 2x16bit format as for above SRC_WIDTH)
```

10400C2Ch - GPU_MEMCOPY_UNKNOWN_FLAG - Memcopy Unknown (R/W)
```
  0      Unknown, R/W
  1-31   Unused (0)
```

Notes...
"The DisplayTransfer registers are only used if FLAGS.bit3=0 and ignored otherwise. The TextureCopy registers are "likewise" only used if FLAGS.bit3=1, and ignored otherwise."

TextureCopy
When FLAGS.bit3=1, the hardware performs a TextureCopy-mode transfer. In this mode, all other bits of that register are ignored (except for FLAGS.bit2, which still needs to be set correctly?), and no format conversions are done.
Instead, it performs a raw data copy from the source to the destination, but with a configurable gap between lines. The total amount of bytes to copy is specified in the size register, and the hardware loops reading lines from the input and writing them to the output until this amount is copied. The "gap" specified in the input/output dimension register is the number of chunks to skip after each "width" chunks of the input/output, and is NOT counted towards the total size of the transfer.

By correctly calculating the input and output gap sizes it is possible to use this functionality to copy arbitrary sub-rectangles between differently-sized framebuffers or textures, which is one of its main uses over a regular no-conversion DisplayTransfer. When copying tiled textures/framebuffers it's important to remember that the contents of a tile are laid out sequentially in memory, and so this should be taken into account when calculating the transfer parameters.

Specifying invalid/junk values for the TextureCopy dimensions can result in the GPU hanging while attempting to process this TextureCopy.

3DS GPU Internal Register Overview

Aliases
It is possible for multiple register (sequential) IDs to correspond to the same register. This is done to leverage the consecutive writing mode for GPU commands, which makes it possible for a single command to write data to multiple sequential register IDs. For example, register IDs 02C1 through 02C8 all correspond to GPUREG_VSH_FLOATUNIFORM_DATAi so that a consecutively writing command based at 02C0 will write its first parameter to GPUREG_VSH_FLOATUNIFORM_INDEX and ever subsequent ones to GPUREG_VSH_FLOATUNIFORM_DATAi

Data Types

```
signed          Signed integer
unsigned        Unsigned integer
floatX.Y.Z      Floating-point number with X sign bits, Y exponent bits,
                  and Z mantissa bits
fixedX.Y.Z      Fixed-point number with X sign bits, Y integer bits,
                  and Z fractional bits
```

float1.5.10 (16bit)
float1.7.12 (20bit) (uncommon, for light attentuation)
float1.7.16 (24bit)
float1.8.23 (32bit)
float1.7.24 (32bit) (uncommon, for viewport)

3DS GPU Internal Registers - Command Lists

Commands (aka GPU registers writes) can be done by manually writing parameters to register 10401000h-10401FFFh, or by using command lists.
With the command lists, the GPU automatically reading parameters and command numbers from memory (and automatically pauses reading when it is busy).

104018E8h - PICA(023Ah) - GPUREG_CMDBUF_ADDR0 (aka entrypoint) (R/W)
104018ECh - PICA(023Bh) - GPUREG_CMDBUF_ADDR1 (aka entrypoint) (R/W)

```
  0      Unused (0)
  1-28   Physical address of command buffer, in 16-byte units
```
Set the physical address of 1st/2nd command buffer.

104018E0h - PICA(0238h) - GPUREG_CMDBUF_SIZE0 (R/W)
104018E4h - PICA(0239h) - GPUREG_CMDBUF_SIZE1 (R/W)
The size value is required to indicate the end of the command list (alternately, if GPUREG_IRQ_AUTOSTOP is enabled, command list execution will also terminate when triggering an IRQ via FINALIZE command; which is usually stored at end of command list).
```
  0      Unused (0)
  1-20   Size of command buffer, in 16-byte units
```
Observe that commands are multiples of 8 bytes (so one may need to append a dummy command for the 16-byte size boundary; unknown if there's a NOP command for that purpose; reportedly FINALIZE command(s) are used for padding, though that could trigger multiple IRQs when GPUREG_IRQ_AUTOSTOP is disabled).

104018F0h - PICA(023Ch) - GPUREG_CMDBUF_JUMP0 (jump to ADDR0) (W)
104018F4h - PICA(023Dh) - GPUREG_CMDBUF_JUMP1 (jump to ADDR0) (W)
```
  0-31   Don't care (writing any value starts CMDBUF execution, at ADDR0/1)
```
Executes the Command List, starting at CMDBUF_ADDR0/1, until reaching CMDBUF_SIZE0/1 (or until FINALIZE+AUTOSTOP).

Unknown why there are two command lists, supposedly they cannot execute simultaneously... maybe the second list is automatically started at the end of the other list?

Command List entries
Each entry in the command list does contain a "Command Header", and one or more parameter words, the entries must be a multiple of 8 bytes.
```
  00h 4   1st Parameter word
  04h 4   Command Header (see below)
  08h 4   2nd Parameter word (if any)
  0Ch 4   3rd Parameter word (if any)
  ... ..  Nth Parameter word (if any)
  xxh 4   Padding (zero) (if above wasn't a multiple of 8 bytes)
```
Command Header:
```
  0-15    Command ID, PICA(0000h..03FFh)
  16      Parameter mask, update parameter bit0-7   (0=Don't change, 1=Write)
  17      Parameter mask, update parameter bit8-15  (0=Don't change, 1=Write)
  18      Parameter mask, update parameter bit16-23 (0=Don't change, 1=Write)
  19      Parameter mask, update parameter bit24-31 (0=Don't change, 1=Write)
  20-27   Number of parameter words-1 (0..FFh = 1..256 parameter words)
  28-30   Unused
  31      Consecutive writing mode (0=Fixed Command number, 1=Increasing)
```

3DS GPU Internal Registers - Finalize Interrupt registers

10401000h+i - PICA(N/A) - GPUREG_IRQ_ACK(0..63) (R/W)
10401040h+i - PICA(0010h..001Fh) - GPUREG_IRQ_REQ(0..63) (R/W)
10401080h+i - PICA(N/A) - GPUREG_IRQ_CMP(0..63) (R/W)
```
  0-7     Request/Compare Byte for IRQ#0   ;\
  8-15    Request/Compare Byte for IRQ#1   ; triggers "P3D" IRQ 2Dh if any
  16-23   Request/Compare Byte for IRQ#2   ; of the request/compare bytes
  ..      ..                               ; are same (and enabled in MASK)
  504-511 Request/Compare Byte for IRQ#63  ;/
```
The ACK registers are basically mirrors of the REQ registers (and ARM11 could write either one, causing IRQ acknowledge or request depending on whether the written value does match the CMP compare bytes). However, there are at least two small differences between ACK and REQ registers:
```
  GPU command list's can only write to GPUREG_IRQ_REQ
```

```
  ARM11 reads from GPUREG_IRQ_REQ cause ARM11 to wait if GPU is busy
```
Normally, CMP contains constant values, and REQ/ACK are changed to same/different values when requesting/releasing interrupt requests.

104010C0h - PICA(N/A) - GPUREG_IRQ_MASK_LOW (R/W)
104010C4h - PICA(N/A) - GPUREG_IRQ_MASK_HIGH (R/W)
```
  0-63   Interrupt Disable flags for IRQ#0..63   (0=Enable, 1=Disable)
```

104010C8h - PICA(N/A) - GPUREG_IRQ_STAT_LOW (R)
104010CCh - PICA(N/A) - GPUREG_IRQ_STAT_HIGH (R)
```
  0-63   Interrupt Compare flags for IRQ#0..63   (0=Mismatch, 1=Match/IRQ)
```

104010D0h - PICA(N/A) - GPUREG_IRQ_AUTOSTOP (R/W)
```
  0      Autostop upon REQ=CMP match  (0=IRQ only, 1=IRQ and Stop cmdlist)
  1-31   Unused (0)
```

104010D4h - PICA(N/A) - GPUREG_FIXED_00010002h (R)
```
  0-31   Unknown, readonly, always 00010002h (some status, or chip id or so?)
```
Maybe related: Port 10400000h contains the same 00010002h readonly value.

Notes
Nintendo uses the IRQ registers to trigger a "FINALIZE" interrupt at end of command list, done as so:
```
  GPUREG_IRQ_MASK_LOW=FFFFFFF0h set upon GPU init  ;\enable first four IRQs
  GPUREG_IRQ_MASK_HIGH=FFFFFFFFh set upon GPU init ;/
  GPUREG_IRQ_AUTOSTOP=1                            ;-autostop cmdlist upon IRQ
  GPUREG_IRQ_CMP(0)=12345678h set upon GPU init    ;-first four compare values
  GPUREG_IRQ_ACK(0)=00000000h set upon GPU init and before cmdlist
  GPUREG_IRQ_REQ(0)=12345678h set via PICA(0010h) FINALIZE at end of cmdlist
```
Due to the 32bit writes, that will actually trigger four byte-matches at once.
Caution:
The GPU command list can write to GPUREG_IRQ_REQ(n), but all other registers in range PICA(0000h..003Fh) are writeable by CPU only, not via command lists.

3DS GPU Internal Registers - Geometry Pipeline registers

10401978h - PICA(025Eh) - GPUREG_PRIMITIVE_CONFIG (R/W)
```
  0-3    Number of vertex shader output map registers - 1        ;see OUTMAP
  8-9    Primitive mode (0=Tri's, 1=Tri-Strip, 2=Tri-Fan, 3=Geometry primitive)
  16     UNKNOWN (R/W)

   __       __ ___          __ __ __
  /\   | /     /| /\  /\        /\ | / .'\    ?????????
 /__\  |/     /_|/__\/__\      /__\|/.'___\   ?????????
 0=Triangles, 1=TriangleStrip, 2=TriangleFan, 3=Geometry primitive
```

1040197Ch - PICA(025Fh) - GPUREG_RESTART_PRIMITIVE (R/W)
```
  0-31   UNKNOWN, usually 1 (writing any value works... but value is R/W)
```
Used to terminate a triangle strip/fan (eg. to start a new strip, instead of appending further vertices to the old strip) (note: the restart must be issued at BEGIN of each strip/fan, not at the end).
Restart occurs on writing any value... but the written value is R/W, so it might have some purpose?

```
_____________________________ Vertex Attribute Arrays _______________________________
```

ATTR_BUF arrays contain vertex attributes (such like coordinates and colors). The GPU can automatically transfer data from these arrays to the shader unit (which does then forward the data to the drawing hardware). There are up to 12 arrays for up to 12 attributes (one can either store multiple attributes in one single array, or store different attributes in separate arrays).

10401804h - PICA(0201h) - GPUREG_ATTR_BUF_FORMAT_LOW (R/W)
10401808h - PICA(0202h) - GPUREG_ATTR_BUF_FORMAT_HIGH (R/W)
```
  0-3    Vertex attribute 0 type/size   ;\these 4bit values contain type/size
  4-7    Vertex attribute 1 type/size   ; type in bit0-1:
  8-11   Vertex attribute 2 type/size   ;  0 = fixed1.0.7  ;Signed byte
  12-15  Vertex attribute 3 type/size   ;  1 = fixed0.0.8  ;Unsigned byte
  16-19  Vertex attribute 4 type/size   ;  2 = fixed1.0.15 ;Signed halfword
  20-23  Vertex attribute 5 type/size   ;  3 = float1.8.23 ;Float32
  24-27  Vertex attribute 6 type/size   ; size in bit2-3:
  28-31  Vertex attribute 7 type/size   ;  0 = X           ;1D
  32-35  Vertex attribute 8 type/size   ;  1 = X,Y         ;2D
  36-39  Vertex attribute 9 type/size   ;  2 = X,Y,Z       ;3D
  40-43  Vertex attribute 10 type/size  ;  3 = X,Y,Z,W     ;4D
  44-47  Vertex attribute 11 type/size  ;/
  48-59  Fixed vertex attribute 0..11 mask flags    (0=Array, 1=Fixed?) ???
  60-63  Number of vertex attributes-1 (0..11 = 1..12 attributes)
```

10401800h - PICA(0200h) - GPUREG_ATTR_BUF_BASE (R/W)
```
  0      Unused (0)
  1-28   Base address for ATTR_BUF's and INDEX_LIST, in 16-byte units
  29-31  Unused (0)
```

1040180Ch+i*0Ch - PICA(0203h+3*(0..11)) - GPUREG_ATTR_BUFi_OFFSET (R/W)
```
  0-27   Vertex array address, with base GPUREG_ATTR_BUF_BASE, in 1-byte units
```
Addressing uses Base+Offset+Index (with Index from FIRST_INDEX or INDEX_LIST):
```
  GPUREG_ATTR_BUF_BASE*10h + GPUREG_ATTR_BUFi_OFFSET*1 + Index*ArrayEntrySize
```

10401810h+i*0Ch - PICA(0204h+3*(0..11)) - GPUREG_ATTR_BUFi_CONFIG_LOW (R/W)
10401814h+i*0Ch - PICA(0205h+3*(0..11)) - GPUREG_ATTR_BUFi_CONFIG_HIGH (R/W)
```
  0-3    Array Component 0      ;\
  4-7    Array Component 1      ; These are mapping the array component(s)
  8-11   Array Component 2      ; to vertex attribute numbers:
  12-15  Array Component 3      ;   00h..0Bh = Vertex attribute 0..11
  16-19  Array Component 4      ;   0Ch      = 4-byte padding
  20-23  Array Component 5      ;   0Dh      = 8-byte padding
  24-27  Array Component 6      ;   0Eh      = 12-byte padding
  28-31  Array Component 7      ;   0Fh      = 16-byte padding
  32-35  Array Component 8      ; Attribute 0..11 do usually refer to vertex
  36-39  Array Component 9      ; shader registers v0..v11 (unless they are
  40-43  Array Component 10     ; renumbered via "PERMUTATION"...?)
  44-47  Array Component 11     ;/
  48-55  Array Entry Size in bytes (0..FFh) ;total size of above component(s)
  56-59  Unused (0)
  60-63  Number of components in this array  (0..12) (0=Disable this array?)
```
Padding: Does "skip" array entries? (maybe useful when having texturing disabled, whilst still having texture info in the array?).
Reportedly there is some alignment issue/feature for padding. And also an alignment issue for components or so; maybe halfword/float32 need to be aligned? There is no alignment restriction for components that consist of bytes (eg. 3x8bit RGB does work).

104018A0h - PICA(0228h) - GPUREG_ATTR_BUF_NUMVERTICES (R/W)
```
  0-31   Number of vertices to render (MUL2 for DRAWELEMENTS)
```
Don't care for FIXEDATTRIB, needed only for ATTR_BUF's.
Counted TWICE for DRAWELEMENTS (once for index-read, and once for array-read)!
XXX are multiple arrays also counted as multiple reads?

104018B8h - PICA(022Eh) - GPUREG_ATTR_BUF_DRAWARRAYS - Increasing Indices (W)
104018BCh - PICA(022Fh) - GPUREG_ATTR_BUF_DRAWELEMENTS - With INDEX_LIST (W)

Writing to either of these registers starts sending the amount of data specified in NUMVERTICES from ATTR_BUF's to the vertex shader.

```
0-31   Don't care (writing any value starts drawing from ATTR_BUF)
```

The only difference between DRAWARRAYS and DRAWELEMENTS are the array indices:

```
DRAWARRAYS uses increasing indices, starting at GPUREG_ATTR_BUF_FIRST_INDEX.
DRAWELEMENTS uses array indices from GPUREG_ATTR_BUF_INDEX_LIST.
```

104018A8h - PICA(022Ah) - GPUREG_ATTR_BUF_FIRST_INDEX (R/W)

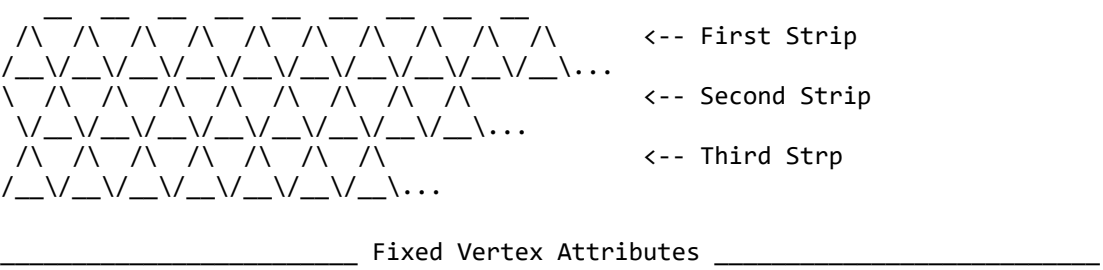
```
0-31   Starting index in ATTR_BUF arrays (usually 0, to start with 1st entry)
```

Used only for for DRAWARRAYS (not used for DRAWELEMENTS, which does instead read indices from the INDEX_LIST).

1040189Ch - PICA(0227h) - GPUREG_ATTR_BUF_INDEX_LIST (R/W)

```
0-27   Index List address, with base GPUREG_ATTR_BUF_BASE, in 1-byte units
31     Index List entry size (0=8bit indices, 1=16bit indices)
```

Used only for DRAWELEMENTS. The index list allows to re-use ATTR_BUF entries for multiple vertices (triangle strips are automatically re-using vertices from the previous triangle, however, with the index list, one can also re-use vertices from the previous strips; which can save about 50% of memory).



Fixed Vertex Attributes

The Fixed Attribute registers can be used in two ways:

```
1) Setting attribute(s) to fixed settings (similar to FLOATUNIFORM
   registers). For example, one could re-use a shader that was
   programmed to support variable colors with fixed colors.
   Unknown if fixed settings are also working for partial attributes
   (eg. 2D graphics with X,Y coordinates and fixed Z,W values).
2) Using [GPUREG_FIXEDATTRIB_INDEX]=0Fh does directly send vertex
   attribute(s) to the shader unit (this could be "easier" than setting
   up ATTR_BUF arrays in memory, however, in practice it is extremely
   uncomfortable (due to the weird data format with 4x24bit float values
   sent in reversed 32bit word order).
```

104018C8h - PICA(0232h) - GPUREG_FIXEDATTRIB_INDEX (W)

```
0-3    Fixed attribute index (00h-0Bh=Index, 0Fh=immediate-mode submission)
```

Selects the index of the fixed attribute to be input with GPUREG_FIXEDATTRIB_DATAi.

104018CCh - PICA(0233h) - GPUREG_FIXEDATTRIB_DATA(0) (HIGH) (W)
104018D0h - PICA(0234h) - GPUREG_FIXEDATTRIB_DATA(1) (MID) (W)
104018D4h - PICA(0235h) - GPUREG_FIXEDATTRIB_DATA(2) (LOW) (W)

```
0-23   float1.7.16, Vertex attribute X
24-47  float1.7.16, Vertex attribute Y
48-71  float1.7.16, Vertex attribute Z
72-95  float1.7.16, Vertex attribute W
```

The above 96bit value is split into three 32bit words, and, weirdly,

```
the word order for writing is HIGH,MID,LOW (aka big endian style)
the byte order per 32bit word is little endian
```

More uncomfortably, fixed attributes must be in 4x24bit float format (with X,Y,Z,W, even when needing only X,Y, and with 24bit float, instead of more comfortable formats like 8bit/16bit integer or 32bit float)
Note: The three DATA registers are mirrors of each other, one could write the HIGH,MID,LOW words all to

DATA0, or to DATA0..2.

_____ Shader/Attribute/Mode Config _____

Below is still rather nebulous, some registers are required to be initialized as described... and some seem to be bogus (somehow having wrong descriptions and/or no function at all).

10401908h - PICA(0242h) - GPUREG_VSH_NUM_ATTR (required) (R/W)
```
  0-3    Number of vertex shader input attributes - 1
```

10401928h - PICA(024Ah) - GPUREG_VSH_OUTMAP_TOTAL1 (required) (R/W)
10401944h - PICA(0251h) - GPUREG_VSH_OUTMAP_TOTAL2 (no function?) (R/W)
```
  0-3    Number of vertex shader output map registers - 1
```
Uh, why are there TWO such registers? Is that for each command buffer? Or for GSH and VSH? Or are that something like total+remaining counts?

104018A4h - PICA(0229h) - GPUREG_GEOSTAGE_CONFIG (unknown purpose) (R/W)
```
  0-1    Geometry shader in use    (0=Not in use, 2=In use)
  8      Drawing triangle elements (0=Not, 1=Drawing triangle elements)
  9      UNKNOWN "0x0" (R/W)
  16-19  UNKNOWN (R/W)
  31     Use reserved geometry shader subdivision (0=Don't use, 1=Use)
```
"When using vertex buffers and drawing elements in triangles mode, bit8 is set to 1, else it is set to 0."
Uh, what means drawing triangle elements... does the gpu even support drawing anything else than triangles?

1040194Ch - PICA(0253h) - GPUREG_GEOSTAGE_CONFIG2 (parts required) (R/W)
```
  0      Function indicator
           with vertex buffers: 0=Draw elements, 1=Draw arrays     <-- blah?
           without: 0=Not inputting, 1=Inputting vertex attribute data  <--?
  8      Drawing triangle elements (0=Not, 1=Drawing triangle elements)
```
"When using vertex buffers, bit0 is set to 1 before drawing arrays, and cleared to 0 immediately after.
When using immediate mode to directly input vertex attribute data, bit0 is set to 1 before inputting vertex attribute data, and cleared to 0 immediately after. While bit0 is set to 1, some register writes outside of the 0x200-0x254 and 0x280-0x2DF ranges may be processed incorrectly.
When using vertex buffers and drawing elements in triangles mode, bit8 is set to 1, else it is set to 0."

10401948h - PICA(0252h) - GPUREG_GSH_MISC0 (R/W)
```
  0-1    Mode?      (0=Normal, 1=WhateverSubdivision?, 2=WhateverParticle?)
  8-22   Something? (0..7FFFh, for the Particle stuff?)
  24     Flag?      (0..1, for the Particle stuff?)
```
Reportedly 00000000h=Normal, 00000001h=Reserved geometry shader subdivision in use, 01004302h=Particle system in use.

10401950h - PICA(0254h) - GPUREG_GSH_MISC1 (R/W)
```
  0-4    Reserved geometry shader subdivision type (2=Loop, 3=Catmull-Clark)
```
"Configures the type of reserved geometry shader subdivision in use. The value is ignored when a subdivision is not in use."

104018B4h - PICA(022Dh) - GPUREG_POST_VERTEX_CACHE_NUM (R/W)
```
  0-7    Number of entries in post-vertex cache (unsigned, usually 04h or 84h)
  8-15   UNKNOWN (R/W)
  16-23  UNKNOWN (R/W)  ;\these two bytes are swapped when reading
  24-31  UNKNOWN (R/W)  ;/(reading mirrors also returns swapped bytes)
```
"Configures the post-vertex cache." Uh, what is that?

104018C4h - PICA(0231h) - GPUREG_VTX_FUNC (W)
```
  0-31   Trigger (0=Idle???, Non-zero=Clear post-vertex cache)
```

"Triggers clearing the post-vertex cache."
Uh, then, maybe should be called "clear cache" instead "vtx func". And what is a "post-vertex"? And is it really clearing anything, drawing seems to work without doing so.

10401910h - PICA(0244h) - GPUREG_VSH_COM_MODE (R/W)
```
  0      Geometry shader configuration enable  (0=Disable, 1=Enable)
```
"Sets whether to use the geometry shader configuration or reuse the vertex shader configuration for the geometry shader shading unit."

10401914h - PICA(0245h) - GPUREG_START_DRAW_FUNC0 (R/W)
```
  0      Mode         (0=Drawing, 1=Configuration)
  1-7    UNKNOWN "0x0" (not R/W)
```
"Related to drawing. When the mode value is set to 1, rendering is not performed properly. When set to 0, changes to the vertex shader configuration registers are not applied correctly. Because of this, it is usually initialized to 1, set to 0 immediately before triggering a draw, and set back to 1 immediately after triggering a draw."

1040190Ch - PICA(0243h) - GPUREG_undoc_1040190Ch (R/W=00000037h)
```
  0-2    UNKNOWN (0..7) (R/W)
  4-6    UNKNOWN (0..3) (R/W)
```
Power-up default is 37h. GPU hangs on several other settings, in some cases also depending on other registers (for example, 00h hangs only when also clearing other undocumented registers).

10401954h - PICA(0255h) - GPUREG_undoc_10401954h (R/W=00000001h)
```
  0      UNKNOWN (0..1) (R/W)
```

3DS GPU Internal Registers - Shader registers

There are four shader units, each having its own set of I/O registers:
```
  10401A00h - PICA(0280h..) - Shader 0 (Geometry Shader, aka GSH)
  10401AC0h - PICA(02B0h..) - Shader 1 (Vertex Shader, aka VSH)
  10401B80h - PICA(02E0h..) - Shader 2 (Unknown purpose)
  10401C40h - PICA(0310h..) - Shader 3 (Unknown purpose)
```
The vertex shader can reportedly use three shader units at once (or all four units, when not using the geometry shader). Even when using multiple units as vertex shader, one does only need to initialize the Shader 1 registers. Unknown if the Shader 2 and 3 registers are having any special purpose for yet unknown special effects (maybe for the undescribed "reserved geometry shader subdivision" feature, or maybe for the "procedural texture" unit, or maybe they are just dummy registers).

```
_____________________________ Input/Output Config _____________________________
```

10401A24h - PICA(0289h) - GPUREG_GSH_INPUTBUFFER_CONFIG (R/W)
10401AE4h - PICA(02B9h) - GPUREG_VSH_INPUTBUFFER_CONFIG (R/W)
10401BA4h - PICA(02E9h) - GPUREG_VSH2_INPUTBUFFER_CONFIG (R/W)
10401C64h - PICA(0319h) - GPUREG_VSH3_INPUTBUFFER_CONFIG (R/W)
```
  0-3    Number of Input vertex attributes - 1
  4-7    Unused (0)
  8      Use reserved geometry shader subdivision (0=Don't use, 1=Use) (uh?)
  9-26   Unused (0)
  27     Whatever, should be 1 for geometry shader (R/W)
  28     Whatever, should be 0                     (R/W)
  29     Whatever, should be 1 for vertex shader   (R/W)
  30     Unused (0)
  31     Whatever, should be 1 for vertex shader   (R/W)
```
Configures the shader unit's input buffer.

10401A2Ch - PICA(028Bh) - GPUREG_GSH_ATTR_PERMUTATION_LOW (R/W)
10401A30h - PICA(028Ch) - GPUREG_GSH_ATTR_PERMUTATION_HIGH (R/W)
10401AECh - PICA(02BBh) - GPUREG_VSH_ATTR_PERMUTATION_LOW (R/W)
10401AF0h - PICA(02BCh) - GPUREG_VSH_ATTR_PERMUTATION_HIGH (R/W)
10401BACh - PICA(02EBh) - GPUREG_VSH2_ATTR_PERMUTATION_LOW (R/W)
10401BB0h - PICA(02ECh) - GPUREG_VSH2_ATTR_PERMUTATION_HIGH (R/W)
10401C6Ch - PICA(031Bh) - GPUREG_VSH3_ATTR_PERMUTATION_LOW (R/W)
10401C70h - PICA(031Ch) - GPUREG_VSH3_ATTR_PERMUTATION_HIGH (R/W)

```
  0-3    Vertex attribute 0 input register index (0..15=v0..v15)
  4-7    Vertex attribute 1 input register index
  8-11   Vertex attribute 2 input register index
  12-15  Vertex attribute 3 input register index
  16-19  Vertex attribute 4 input register index
  20-23  Vertex attribute 5 input register index
  24-27  Vertex attribute 6 input register index
  28-31  Vertex attribute 7 input register index
  32-35  Vertex attribute 8 input register index
  36-39  Vertex attribute 9 input register index
  40-43  Vertex attribute 10 input register index
  44-47  Vertex attribute 11 input register index
  48-51  Vertex attribute 12 input register index  ;\uh, how does one
  52-55  Vertex attribute 13 input register index  ; set attr 12-15?
  56-59  Vertex attribute 14 input register index  ; maybe FROM
  60-63  Vertex attribute 15 input register index  ;/geometry shader?
```

Sets the shader unit input register index which will correspond to each attribute contained by the input buffer (which in the case of geometry shaders is the vertex shader output buffer (uh, that says that there is ONE vertex shader, and MULTIPLE geometry shaders?)) for attributes 0 through 15.
For example, having LOW.bit0-3 set to 5 means that, in the shader program, v5 will contain the input buffer's 1st attribute.
For example, having HIGH.bit0-3 set to 5 means that, in the shader program, v5 will contain the input buffer's 9th attribute.
somewhat semantic?
Uh, ATTR_PERMUTATION is a scary homebrew slang for ATTR_IDs?

10401A34h - PICA(028Dh) - GPUREG_GSH_OUTMAP_MASK (R/W)
10401AF4h - PICA(02BDh) - GPUREG_VSH_OUTMAP_MASK (R/W)
10401BB4h - PICA(02EDh) - GPUREG_VSH2_OUTMAP_MASK (R/W)
10401C74h - PICA(031Dh) - GPUREG_VSH3_OUTMAP_MASK (R/W)

```
  0      Output register o0  (0=Disable, 1=Enable)
  1      Output register o1  (0=Disable, 1=Enable)
  2      Output register o2  (0=Disable, 1=Enable)
  3      Output register o3  (0=Disable, 1=Enable)
  4      Output register o4  (0=Disable, 1=Enable)
  5      Output register o5  (0=Disable, 1=Enable)
  6      Output register o6  (0=Disable, 1=Enable)
  7      Output register o7  (0=Disable, 1=Enable) (vertex shader only, uh?)
  8      Output register o8  (0=Disable, 1=Enable) (vertex shader only, uh?)
  9      Output register o9  (0=Disable, 1=Enable) (vertex shader only, uh?)
  10     Output register o10 (0=Disable, 1=Enable) (vertex shader only, uh?)
  11     Output register o11 (0=Disable, 1=Enable) (vertex shader only, uh?)
  12     Output register o12 (0=Disable, 1=Enable) (vertex shader only, uh?)
  13     Output register o13 (0=Disable, 1=Enable) (vertex shader only, uh?)
  14     Output register o14 (0=Disable, 1=Enable) (vertex shader only, uh?)
  15     Output register o15 (0=Disable, 1=Enable) (vertex shader only, uh?)
  16-26  UNKNOWN (R/W)
  16-31  UNKNOWN "0x0"  (uh, but only bit27-31 are always 0)
```

Toggles a shader unit's output registers.
See also: GPUREG_SH_OUTMAP_Oi

Uniform Registers

Uniform registers can contain general purpose constants. That is, they are usually kept set to constant values throughout sending multiple vertices (unlike position/texture coordinates or other vertex attributes that differ on each vertex).

10401A00h - PICA(0280h) - GPUREG_GSH_BOOLUNIFORM (R/W)
10401AC0h - PICA(02B0h) - GPUREG_VSH_BOOLUNIFORM (R/W)
10401B80h - PICA(02E0h) - GPUREG_VSH2_BOOLUNIFORM (R/W)
10401C40h - PICA(0310h) - GPUREG_VSH3_BOOLUNIFORM (R/W)

```
  0-15   Boolean register b0..b15 value  (0=False, 1=True)
  16-27  For GSH: UNKNOWN (0..0FFFh) (R/W)  ;\unknown, but R/W mask resembles
  16-24  For VSH: UNKNOWN (0..01FFh) (R/W)  ;/GPUREG_xSH_ENTRYPOINT !
  16-31  UNKNOWN "0x7FFF"  (uh, but bit28-31 are always 0?) (for VSH: 25-31)
```

The sixteen 1bit boolean registers (B0..B15) can be used by conditional shader opcodes. Uh, elsewhere it's claimed that there are only eight bool registers?
Bit15 seems to be automatically set after executing shader code?
Bit16 and up seem to contain an opcode address?

10401A04h+i*4 - PICA(0281h+i) - GPUREG_GSH_INTUNIFORM_I0..I3 (R/W)
10401AC4h+i*4 - PICA(02B1h+i) - GPUREG_VSH_INTUNIFORM_I0..I3 (R/W)
10401B84h+i*4 - PICA(02E1h+i) - GPUREG_VSH2_INTUNIFORM_I0..I3 (R/W)
10401C44h+i*4 - PICA(0311h+i) - GPUREG_VSH3_INTUNIFORM_I0..I3 (R/W)

```
  0-7    unsigned, Integer register ii X value  aka INT.x  Loop count-1
  8-15   unsigned, Integer register ii Y value  aka INT.y  Loop starting index
  16-23  unsigned, Integer register ii Z value  aka INT.z  Loop index step
  24-31  Unused (0)
```

Each of the four integer registers (I0..I3) contains three 8bit integer values (X,Y,Z). The integers are used to initalize loop counters and indexing.

10401A40h - PICA(0290h) - GPUREG_GSH_FLOATUNIFORM_INDEX (W)
10401B00h - PICA(02C0h) - GPUREG_VSH_FLOATUNIFORM_INDEX (W)
10401BC0h - PICA(02F0h) - GPUREG_VSH2_FLOATUNIFORM_INDEX (W)
10401C80h - PICA(0320h) - GPUREG_VSH3_FLOATUNIFORM_INDEX (W)

```
  0-7    Transfer index for FLOATUNIFORM_DATA writes (0..95 = C0..C95)
  31     Transfer mode                  (0=float1.7.16, 1=float1.8.23)
```

10401A44h+i*4 - PICA(0291h+i) - GPUREG_GSH_FLOATUNIFORM_DATA(0..7) (W)
10401B04h+i*4 - PICA(02C1h+i) - GPUREG_VSH_FLOATUNIFORM_DATA(0..7) (W)
10401BC4h+i*4 - PICA(02F1h+i) - GPUREG_VSH2_FLOATUNIFORM_DATA(0..7) (W)
10401C84h+i*4 - PICA(0321h+i) - GPUREG_VSH3_FLOATUNIFORM_DATA(0..7) (W)

Each of the 96 float uniform registers (C0..C95) contains four float values (X,Y,Z,W). The floats can contain general purpose constants, colors, vectors, or matrices (eg. when using four such registers as matrix with 4x4 float values).
"The data format which should be written to it depends on the transfer mode set with GPUREG_SH_FLOATUNIFORM_INDEX. These registers function as a FIFO queue: after each time a 4-component uniform register is successfully set, the target register index is incremented, meaning that groups of uniforms with contiguous register IDs can be set with only one initial write to GPUREG_SH_FLOATUNIFORM_INDEX."
float24 (float1.7.16) transfer mode:
"Data should be sent by writing three words which are the concatenation of the float24 value of the uniform register's 4 components, in the "reverse order". Assuming each letter corresponds to 4 bits, the format becomes:"

```
  (uh, is that the aforementioned "reverse order", or is it unreversed here?)
  1st word  : ZZWWWWWW  ;uh, reverse of GPUREG_FIXEDATTRIB_DATA ??
  2nd word  : YYYYZZZZ
  3rd word  : XXXXXXYY
```

float32 (float1.8.23) transfer mode:
"Data should be sent by writing four words which are each the float32 value of the uniform register's 4 components, in the reverse order... uh, who/what/how is that reverse order??? probably alike as shown below (or reverse thereof?):"

```
  1st word  : WWWWWWWW
  2nd word  : ZZZZZZZZ
  3rd word  : YYYYYYYY
  4th word  : XXXXXXXX
```

```
 _____________________________ Opcodes and Opdesc's _________________________________
```

10401A28h - PICA(028Ah) - GPUREG_GSH_ENTRYPOINT (R/W)
10401AE8h - PICA(02BAh) - GPUREG_VSH_ENTRYPOINT (R/W)
10401BA8h - PICA(02EAh) - GPUREG_VSH2_ENTRYPOINT (R/W)
10401C68h - PICA(031Ah) - GPUREG_VSH3_ENTRYPOINT (R/W)

```
  0-15   Code entry point offset, in 32-bit words
  16-31  0x7FFF      ;maybe max/end address for error-checks?
```

uh, but, for GSH, R/W mask is 0FFF0FFFh.
uh, but, for VSH, R/W mask is 01FF01FFh.

10401A6Ch - PICA(029Bh) - GPUREG_GSH_CODETRANSFER_INDEX (W)
10401B2Ch - PICA(02CBh) - GPUREG_VSH_CODETRANSFER_INDEX (W)
10401BECh - PICA(02FBh) - GPUREG_VSH2_CODETRANSFER_INDEX (W)
10401CACh - PICA(032Bh) - GPUREG_VSH3_CODETRANSFER_INDEX (W)

```
  0-11   Target shader code offset, in WHAT units? 4-byte maybe?
```

Offset at which upcoming shader code data transferred through GPUREG_SH_CODETRANSFER_DATAi should be written.

10401A70h+i*4 - PICA(029Ch+i) - GPUREG_GSH_CODETRANSFER_DATA(0..7) (W)
10401B30h+i*4 - PICA(02CCh+i) - GPUREG_VSH_CODETRANSFER_DATA(0..7) (W)
10401BF0h+i*4 - PICA(02FCh+i) - GPUREG_VSH2_CODETRANSFER_DATA(0..7) (W)
10401CB0h+i*4 - PICA(032Ch+i) - GPUREG_VSH3_CODETRANSFER_DATA(0..7) (W)

```
  0-31   Shader instruction data
```

Used to transfer shader code data. These registers behave as a FIFO queue: each write to these registers writes the provided value to the GPU shader code memory bank at the offset initially set by GPUREG_SH_CODETRANSFER_INDEX. The offset in question is incremented after each write to this register.

10401A3Ch - PICA(028Fh) - GPUREG_GSH_CODETRANSFER_END (W)
10401AFCh - PICA(02BFh) - GPUREG_VSH_CODETRANSFER_END (W)
10401BBCh - PICA(02EFh) - GPUREG_VSH2_CODETRANSFER_END (W)
10401C7Ch - PICA(031Fh) - GPUREG_VSH3_CODETRANSFER_END (W)

```
  0-31   Don't care (writing any value works)
```

Unknown what that does. It seems to be needed at least once after first code upload (but is usually done after each shader code upload). Maybe enables code execution, or invalidates some cache?

10401A94h - PICA(02A5h) - GPUREG_GSH_OPDESCS_INDEX (W)
10401B54h - PICA(02D5h) - GPUREG_VSH_OPDESCS_INDEX (W)
10401C14h - PICA(0305h) - GPUREG_VSH2_OPDESCS_INDEX (W)
10401CD4h - PICA(0335h) - GPUREG_VSH3_OPDESCS_INDEX (W)

```
  0-11   Target shader operand descriptor offset, in WHAT units?
```

Uh, why 12bit wide? There are only max 128 descriptors per shader.
Offset at which upcoming shader operand descriptor data transferred through GPUREG_SH_OPDESCS_DATAi should be written.

10401A98h+i*4 - PICA(02A6h+i) - GPUREG_GSH_OPDESCS_DATA(0..7) (W)

10401B58h+i*4 - PICA(02D6h+i) - GPUREG_VSH_OPDESCS_DATA(0..7) (W)
10401C18h+i*4 - PICA(0306h+i) - GPUREG_VSH2_OPDESCS_DATA(0..7) (W)
10401CD8h+i*4 - PICA(0336h+i) - GPUREG_VSH3_OPDESCS_DATA(0..7) (W)
```
  0-31   Shader operand descriptor data
```
Used to transfer shader operand descriptor data. These registers behave as a FIFO queue: each write to these registers writes the provided value to the GPU shader operand descriptor memory bank at the offset initially set by GPUREG_SH_OPDESCS_INDEX. The offset in question is incremented after each write to this register.

3DS GPU Internal Registers - Rasterizer registers

```
_____________________________ Viewport and Scissor _____________________________
```

10401104h - PICA(0041h) - GPUREG_VIEWPORT_V_SCALE (3DS: 240/2) (R/W)
1040110Ch - PICA(0043h) - GPUREG_VIEWPORT_H_SCALE (3DS: 320/2 or 400/2) (R/W)
```
  0-23   float1.7.16, size/2
```
Used to scale the vertex coordinates (usually in range -1.0 to +1.0) to screen coordinates (in pixel units). Should be set to "H=width/2" and "V=height/2". Normally that would be the screen size (or for smaller windows, the scissor area size).

10401108h - PICA(0042h) - GPUREG_VIEWPORT_V_STEP (3DS: 2/240) (R/W)
10401110h - PICA(0044h) - GPUREG_VIEWPORT_H_STEP (3DS: 2/320 or 2/400) (R/W)
```
  0-31   float1.7.24, 2/size
```
Used as stepping offsets when interpolating colors (and textures?) between vertex coordinates. Should be set to same settings as in WIDTH and HEIGHT registers, but with "inverse" division, ie. "H=2/width" and "V=2/height" (instead of "width/2" and "height/2").
For the odd floating point format, one could adjust one of the more common formats (assuming sign=0 for the latter two cases):
```
  (float24)*100h                 ;convert float1.7.16 to float1.7.24
  (float32)*2-40000000h          ;convert float1.8.23 to float1.7.24
  (float64)/10000000h-0C0000000h  ;convert float1.11.52 to float1.7.24
```
or it might also work to directly use screen coordinates for vertices (and set all viewport registers to 1.0).

10401194h - PICA(0065h) - GPUREG_SCISSORTEST_MODE (R/W)
```
  0-1    Mode (0=Disable, 1=Render Outside, 3=Render Inside, 2=Same as 0)
```
Allows to disable the scissor feature (ie. to render to render to whole framebuffer), or to select whether to render only inside, or only outside of the scissor rectangle. This can be useful if the framebuffer is divided into smaller windows.

10401198h - PICA(0066h) - GPUREG_SCISSORTEST_POS1 (R/W)
1040119Ch - PICA(0067h) - GPUREG_SCISSORTEST_POS2 (R/W)
```
  0-9    Vertical Position   (unsigned, 0..3FFh pixels)  ;0..240-1
  16-25  Horizontal Position (unsigned, 0..3FFh pixels)  ;0..320-1 or 0..400-1
```
Defines the scissor area, ranging from POS1=origin to POS2=origin+size-1 (whereas, coordinate 0,0 is lower-right).

104011A0h - PICA(0068h) - GPUREG_VIEWPORT_XY (R/W)
```
  0-9    Vertical Position   (signed, -200h..+1FFh pixels)
  16-25  Horizontal Position (signed, -200h..+1FFh pixels)
```
Default is 0,0 (with coordinate 0,0 being at center of framebuffer). Nonzero values can cause polygons to wrap at the framebuffer edges (unless scissor test is used, with position plus scissor size being within framebuffer dimensions).

```
_____________________________________ Misc _____________________________________
```

10401100h - PICA(0040h) - GPUREG_FACECULLING_CONFIG (R/W)

```
  0-1    Culling (0=Show Front+Back, 1=Show Back, 2=Show Front, 3=Same as 0)
```
Selects whether to show the polygon front/back sides. Normally only the front side is needed (and it's thus faster not to render the back side).
Note: The hardware detects front/back by checking whether the triangle's coordinates are arranged clockwise or anti-clockwise on screen.

1040111Ch - PICA(0047h) - GPUREG_FRAGOP_CLIP (R/W)
```
  0      Enable clipping plane(s?) (0=Disable, 1=Enable)
```

10401120h+i*4 - PICA(0048h+i) - GPUREG_FRAGOP_CLIP_DATAi (i=0..3) (R/W)
```
  0-23   float1.7.16, Clipping plane coefficient i
```
"Used to configure clipping plane coefficients."
Note: There is a similar poorly described feature in OpenGL, but it's unknown what it does and how to use it. It does somehow add another clipping plane (additionally to the left/right, top/bottom, and near/far clipping plane).

10401134h - PICA(004Dh) - GPUREG_DEPTHMAP_SCALE (R/W)
```
  0-23   float1.7.16, Near - Far             ;default would be Z=-0.99999999
```
Used to configure the depth range scale.
Uh, what means map, hmmm, maybe it just means "bitmap" with depth values?
Maybe defines the visible depth range? In Z or W units? But why called scale?
Maybe also "scales" float values to 24bit integers in depth buffer?

10401138h - PICA(004Eh) - GPUREG_DEPTHMAP_OFFSET (R/W)
```
  0-23   float1.7.16, Near + Polygon Offset  ;default would be Z=+0.0
```
Used to configure the depth range bias.
Probably defines near clipping plane, in Z or W units? Unknown what "Polygon Offset" refers to.

104011B4h - PICA(006Dh) - GPUREG_DEPTHMAP_ENABLE (R/W)
```
  0      Enable depth range, uh what? (0=Disable/What??, 1=Enable/Default)
```

```
_____________________________________ Early Depth _____________________________________
```

Early Depth is something about drawing only depth values (without computing and drawing colors/textures) for pre-checking if the polygon is visible... if so, something is probably required to actually draw the color/texture part...?

10401184h - PICA(0061h) - GPUREG_EARLYDEPTH_FUNC (R/W)
```
  0-1    Early depth function (0=GreaterEqual, 1=Greater, 2=LessEqual, 3=Less)
```
Configures the early depth test function.

10401188h - PICA(0062h) - GPUREG_EARLYDEPTH_TEST1 (R/W)
10401460h - PICA(0118h) - GPUREG_EARLYDEPTH_TEST2 (R/W)
```
  0      Enable early depth test    (0=Disable, 1=Enable)
```

1040118Ch - PICA(0063h) - GPUREG_EARLYDEPTH_CLEAR (W)
```
  0      Trigger (0=Idle???, 1=Clear) (W)   ;read=Always 0
```
Triggers clearing the early depth data. Whatever that means, probably something for restarting a new early depth test (whatever, it does NOT memfill the DEPTHBUFFER).

104011A8h - PICA(006Ah) - GPUREG_EARLYDEPTH_DATA (R/W)
```
  0-23   Early depth Clear value (unsigned)
```

```
_____________________________________ Shader Output _____________________________________
```

1040113Ch - PICA(004Fh) - GPUREG_SH_OUTMAP_TOTAL (R/W)
```
  0-2    Number of shader output attributes (1..7)
```

10401140h+i*4 - PICA(0050h+i) - GPUREG_SH_OUTMAP_Oi (i=0..6) (R/W)

```
  0-4    Semantic for the x component of the register (0..1Fh, see below)
  8-12   Semantic for the y component of the register (0..1Fh, see below)
  16-20  Semantic for the z component of the register (0..1Fh, see below)
  24-28  Semantic for the w component of the register (0..1Fh, see below)
```

These registers map components of the corresponding vertex shader output register (o0..o6) to specific fixed-function semantics.
Semantic values:

```
  00h = position.x         ;\
  01h = position.y         ; Vertex Position
  02h = position.z         ;
  03h = position.w         ;/
  04h = normquat.x         ;\
  05h = normquat.y         ; Quaternion specifying the normal/tangent frame
  06h = normquat.z         ; (for fragment lighting)
  07h = normquat.w         ;/
  08h = color.r            ;\
  09h = color.g            ; Vertex color
  0Ah = color.b            ;
  0Bh = color.a            ;/
  0Ch = texcoord0.u        ;\Texture coordinates for texture 0
  0Dh = texcoord0.v        ;/
  0Eh = texcoord1.u        ;\Texture coordinates for texture 1
  0Fh = texcoord1.v        ;/
  10h = texcoord0.w        ;-Texture coordinate.w for texture 0
  12h = view.x             ;\
  13h = view.y             ; View vector (for fragment lighting)
  14h = view.z             ;/
  16h = texcoord2.u        ;\Texture coordinates for texture 2
  17h = texcoord2.v        ;/
  1Fh = Unused component   ;-for unused components of the output register
```

Observe that the X,Y,Z,W positions require Z,W even for simple 2D graphics (Z should be -1.0..+0.0 to pass near/far clipping, W should be +1.0 to avoid perspective division of the X,Y,Z values).

104011BCh - PICA(006Fh) - GPUREG_SH_OUTATTR_CLOCK (R/W)

```
  0      'position.z' present                   (0=Absent, 1=Present)
  1      'color' component present              (0=Absent, 1=Present)
  8      'texcoord0' component present          (0=Absent, 1=Present)
  9      'texcoord1' component present          (0=Absent, 1=Present)
  10     'texcoord2' component present          (0=Absent, 1=Present)
  11-15  UNKNOWN (R/W)
  16     'texcoord0.w' present                  (0=Absent, 1=Present)
  17     UNKNOWN (R/W)
  24     'normquat' or 'view' component present (0=Absent, 1=Present)
```

Controls the clock supply to parts relating to certain attributes.
somewhat semantic?

10401190h - PICA(0064h) - GPUREG_SH_OUTATTR_MODE (R/W)

```
  0      Use texture coordinates (0=Don't use, 1=Use)
```

Configures the shader output attribute mode, uh?

Undocumented

10401114h - PICA(0045h) - GPUREG_undoc_10401114h (R/W=00FFFFFFh)
10401118h - PICA(0046h) - GPUREG_undoc_10401118h (R/W=00FFFFFFh)

```
  0-23   UNKNOWN (0..FFFFFFh) (R/W)
```

10401130h - PICA(004Ch) - GPUREG_undoc_10401130h (R/W=00000001h)

```
  0      UNKNOWN (0..1) (R/W)
```

10401160h - PICA(0058h) - GPUREG_undoc_10401160h (R/W=00000101h)
```
  0      Flat Shading (0=Interpolate Colors, 1=Flat Uni-Color)
  8      UNKNOWN (0..1) (R/W)
```
The Flat Uni-Color mode uses only the third (last) color of each triangle.
Unknown if that does also apply to colors calculated from light vectors.

10401164h - PICA(0059h) - GPUREG_undoc_10401164h (R/W=00000001h)
```
  0      UNKNOWN (0..1) (R/W)
```

10401180h - PICA(0060h) - GPUREG_undoc_10401180h (R/W=00000301h)
```
  0      UNKNOWN (0..1) (R/W)
  8-9    Draw pixel(s) (0=All/Normal, 1=Each 2nd, 2=Each 4th, 3=Same as 0)
```
Allows to draw only each 2nd/4th pixel within 2x2 pixel cells (resulting in striped/dotted output).

104011A4h - PICA(0069h) - GPUREG_undoc_104011A4h (R/W=FFFF0001h)
```
  0      UNKNOWN (0..1) (R/W)
  16-31  UNKNOWN (0..FFFFh) (R/W)
```

104011ACh - PICA(006Bh) - GPUREG_undoc_104011ACh (R/W=00000FFFh)
```
  0-11   UNKNOWN (0..FFFh) (R/W)
```

104011B0h - PICA(006Ch) - GPUREG_undoc_104011B0h (DANGER, hangs when reading)
```
  0-31   UNKNOWN/DANGER, hangs when reading (maybe a fifo, hangs when empty?)
```

```
_____________________________________ Status _____________________________________
```

10401168h - PICA(005Ah) - GPUREG_STAT_NUM_VERTICES_RECEIVED (R)
1040116Ch - PICA(005Bh) - GPUREG_STAT_NUM_TRIANGLES_RECEIVED (R)
10401170h - PICA(005Ch) - GPUREG_STAT_NUM_TRIANGLES_DISPLAYED (R)
Indicates the number of processed vertices/triangles, the third counter does increment only on actually displayed triangles (ie. when having passed the front/back check) (and maybe further checks, like screen/scissor area).
```
  0-xx?  Status counter (incrementing)
```
Note: Reading from these registers (or any other GPU registers at 10401040h..1040107Fh or 10401100h..10401FFFh) does cause ARM11 to wait until rendering has finished.

3DS GPU Internal Registers - Framebuffer registers

10401400h - PICA(0100h) - GPUREG_COLOR_OPERATION (R/W)
```
  0-1    Fragment operation mode    (0=Default, 1=Gas, 2=?, 3=Shadow)
  8      Blend mode                 (0=Logic op, 1=Blend)
  16-23  Unused, reportedly E4h ?   (isn't R/W... maybe write-only?)
  24     Render only each 2nd line  (0=All lines, 1=Only each 2nd line)
  25     Render nothing?            (0=Render, 1=Nothing)
```

10401404h - PICA(0101h) - GPUREG_BLEND_FUNC (R/W)
```
  0-2    RGB equation   (0=Add, 1=Sub, 2=ReverseSub, 3=Min, 4=Max, 5/6/7=Add)
  8-10   Alpha equation (0=Add, 1=Sub, 2=ReverseSub, 3=Min, 4=Max, 5/6/7=Add)
  16-19  RGB source blending function         (00h..0Fh, see below)
  20-23  RGB destination blending function    (00h..0Fh, see below)
  24-27  Alpha source blending function       (00h..0Fh, see below)
  28-31  Alpha destination blending function  (00h..0Fh, see below)
```
Blending Function values:
```
  0h = Zero
  1h = One
  2h = Source color
  3h = One minus source color
```

```
  4h = Destination color
  5h = One minus destination color
  6h = Source alpha                ;<-- use for source func's?
  7h = One minus source alpha      ;<-- use for dest func's?
  8h = Destination alpha
  9h = One minus destination alpha
  Ah = Constant color              ;<-- aka GPUREG_BLEND_COLOR maybe?
  Bh = One minus constant color
  Ch = Constant alpha
  Dh = One minus constant alpha
  Eh = Source alpha saturate       ;<-- uh?
  Fh = ?
```

Unknown what source/dest means, maybe source=polygon, dest=colorbuf?
Unknown if there's a way to disable blending (other than using ADD+Zero).
Selecting things like "Source alpha" as RGB func seems to mean "color*alpha"?
Unknown if this (or other registers) allow anti-aliasing.

10401408h - PICA(0102h) - GPUREG_LOGIC_OP (R/W)

```
  0-3    Logic op (0..0Fh, see below)
```

Logic op values:

```
  0h = Clear
  1h = AND
  2h = Reverse AND
  3h = Copy
  4h = Set
  5h = Inverted copy
  6h = Noop
  7h = Invert
  8h = NAND
  9h = OR
  Ah = NOR
  Bh = XOR
  Ch = Equivalent
  Dh = Inverted AND
  Eh = Reverse OR
  Fh = Inverted OR
```

Uh, what does that crap mean, and when is it used what for?

1040140Ch - PICA(0103h) - GPUREG_BLEND_COLOR (R/W)

```
  0-7    Red    (unsigned, 00h..FFh)
  8-15   Green  (unsigned, 00h..FFh)
  16-23  Blue   (unsigned, 00h..FFh)
  24-31  Alpha  (unsigned, 00h..FFh)
```

10401410h - PICA(0104h) - GPUREG_FRAGOP_ALPHA_TEST (R/W)

```
  0      Alpha Testing Enable     (0=Disable, 1=Enable)
  4-6    Alpha Testing Function   (0-7, see below)
  8-15   Alpha Reference value    (00h..FFh, unsigned)
```

Alpha Testing Function values:

```
  0h = Never
  1h = Always
  2h = Equal
  3h = Not equal
  4h = Less than
  5h = Less than or equal
  6h = Greater than
  7h = Greater than or equal
```

10401414h - PICA(0105h) - GPUREG_STENCIL_TEST (R/W)

```
  0      Stencil Testing Enable    (0=Disable, 1=Enable)
  4-6    Stencil Testing Function  (0-7, see below)
  8-15   Buffer mask               (unsigned, 00h..FFh)
```

```
  16-23  Reference value          (signed, -80h..+7Fh)
  24-31  Mask                     (unsigned, 00h..FFh)
```
Stencil Testing Function values:
```
  0h = Never
  1h = Always
  2h = Equal
  3h = Not equal
  4h = Less than
  5h = Less than or equal
  6h = Greater than
  7h = Greater than or equal
```

10401418h - PICA(0106h) - GPUREG_STENCIL_OP (R/W)
```
  0-2    Stencil Fail operation   (0..7, see below)
  4-6    Stencil Z-fail operation (0..7, see below)
  8-10   Stencil Z-pass operation (0..7, see below)
```
Operation values:
```
  0h = Keep
  1h = Zero
  2h = Replace
  3h = Increment
  4h = Decrement
  5h = Invert
  6h = Increment and wrap
  7h = Decrement and wrap
```

1040141Ch - PICA(0107h) - GPUREG_DEPTH_COLOR_MASK (R/W)
```
  0      Depth test enable   (0=Disable, 1=Enable)  ;\Depth testing
  4-6    Depth function      (0..7, see below)      ;/
  8      Red write enable    (0=Disable, 1=Enable)  ;\
  9      Green write enable  (0=Disable, 1=Enable)  ; Color Buffer writing
  10     Blue write enable   (0=Disable, 1=Enable)  ;
  11     Alpha write enable  (0=Disable, 1=Enable)  ;/
  12     Depth write enable  (0=Disable, 1=Enable)  ;-Depth Buffer writing
```
Depth function values:
```
  0h = Never
  1h = Always (same as when depth test is disabled)
  2h = Equal
  3h = Not equal
  4h = Less than
  5h = Less than or equal
  6h = Greater than
  7h = Greater than or equal
```

10401440h - PICA(0110h) - GPUREG_RENDERBUFFER_INVALIDATE (forget cache) (W)
10401444h - PICA(0111h) - GPUREG_RENDERBUFFER_FLUSH (cache writeback) (W)
```
  0      Trigger (0=No change, 1=Trigger) (W)
  1-31   Unused (0)
```
INVALIDATE: Forgets Color/Depth Cache contents (should be used before changing buffer base addresses, or before externally MEMFILL'ing the Color/Depth buffers).
FLUSH: Copies Color/Depth Cache to memory (should be used before MEMCOPY'ing the Color Buffer to LCD Framebuffer).

10401448h - PICA(0112h) - GPUREG_COLORBUFFER_READING - Allow Read (R/W)
1040144Ch - PICA(0113h) - GPUREG_COLORBUFFER_WRITING - Allow Write (R/W)
```
  0-3    Allow read/write (00h=Disable, 0Fh=Enable, 01h..0Eh=Same as 0Fh?)
```
Write disable prevents polygon drawing. Unknown what happens on read disable.
Note: Disable stops the counters at 10400098h/1040009Ch.

10401450h - PICA(0114h) - GPUREG_DEPTHBUFFER_READING - Allow READ (R/W)
10401454h - PICA(0115h) - GPUREG_DEPTHBUFFER_WRITING - Allow WRITE (R/W)

```
0      Allow stencil read/write (0=Disable, 1=Enable)
1      Allow depth read/write   (0=Disable, 1=Enable)
```

Note: Disable stops the counters at 10400090h/10400094h.

10401458h - PICA(0116h) - GPUREG_DEPTHBUFFER_FORMAT (R/W)

```
0-1    Format (0=16bitDepth, 1=?, 2=24bitDepth, 3=24bitDepth+8bitStencil)
```

Configures the depth buffer data format.

1040145Ch - PICA(0117h) - GPUREG_COLORBUFFER_FORMAT (R/W)

```
0-1    Pixel size (0=16bitColor, 2=32bitColor, 1/3=?)
16-18  Format     (0=RGBA8/Gas, 2=RGB5A1, 3=RGB565, 4=RGBA4, 1/5/6/7=?)
```

Configures the color buffer data format. Color components are laid out in reverse byte order in memory, with the most significant bits used first. uh, maybe that refers to two 16bit pixels within one 32bit word?

1040146Ch - PICA(011Bh) - GPUREG_RENDERBUFFER_BLOCK32 (R/W)

```
0      Render buffer block size (0=Normal/8x8 pixels, 1=32x32 pixels)
```

Should be set to 8x8 (the 32x32 pixel mode can be reportedly used with GPU_MEMCOPY_FLAGS.bit16=1, but 32x32 won't match up with the 400x240 and 320x240 pixel 3DS screens)

10401470h - PICA(011Ch) - GPUREG_DEPTHBUFFER_LOC (R/W)
10401474h - PICA(011Dh) - GPUREG_COLORBUFFER_LOC (R/W)

```
0-2    UNKNOWN, seems to have no effect (R/W)
3-27   Buffer physical address, in 64-byte units
```

These registers configure the depth/color buffer physical addresses. The buffers are reportedly divided into 8x8 pixel blocks (or optionally 32x32 pixel), that tiled format does supposedly improve cache hits when drawing small polygons.
The color buffer can contain RGBA values. After rendering, these should be copied to LCD framebuffer(s) using GPU_MEMCOPY registers (which can convert the color format, and convert tiled memory to linear order.
The depth buffer can contain Depth and Stencil values. Unknown if the Depth values are float or integer values, and unknown if they are Z or W values (or if either one is selectable).

10401478h - PICA(011Eh) - GPUREG_RENDERBUFFER_DIM_0 (R/W)
104011B8h - PICA(006Eh) - GPUREG_RENDERBUFFER_DIM_1 (R/W)

```
0-10   Height     (unsigned, 0..7FFh) (usually 240 on 3DS)
12-21  Width - 1  (unsigned, 0..3FFh) (usually 320-1 or 400-1 on 3DS)
24     Negate Y-coordinates  (0=Mirror; horizontally on 3DS, 1=Default)
```

Configures the size of the COLORBUFFER and DEPTHBUFFER. The DIM_0 and DIM_1 registers should be usually set to the same value (although DIM_0 seems to be needed, DIM_1 seems to have no effect).

10401434h - PICA(010Dh) - GPUREG_undoc_10401434h (R/W=00000001h)

```
0      UNKNOWN (0..1) (R/W)
```

10401438h - PICA(010Eh) - GPUREG_undoc_10401438h (R/W=FFFFFFFFh)
1040143Ch - PICA(010Fh) - GPUREG_undoc_1040143Ch (R/W=FFFFFFFFh)

```
0-31   UNKNOWN (0..FFFFFFFFh) (R/W)
```

10401464h - PICA(0119h) - GPUREG_undoc_10401464h (R/W=FFFFFFFFh)
10401468h - PICA(011Ah) - GPUREG_undoc_10401468h (R/W=FFFFFFFFh)

```
0-31   UNKNOWN (0..FFFFFFFFh) (R/W)
```

1040147Ch - PICA(011Fh) - GPUREG_undoc_1040147Ch (R/W=7FFFFFFFh)

```
0-30   UNKNOWN (0..7FFFFFFFh) (R/W)
```

Power-up default is 00020200h, should be set to 00010140h. GPU hangs when set to 7FFFFFFFh. Value 1FFFFFFFh can cause a few dirt pixels upon polygon rendering. Guess: Maybe some PLL timing or memory

waitstate/cache stuff?

10401480h - PICA(0120h) - GPUREG_GAS_LIGHT_XY - Planar Shading (R/W)
10401484h - PICA(0121h) - GPUREG_GAS_LIGHT_Z - View Shading (R/W)
```
  0-7    shading minimum intensity   (unsigned, 00h..FFh)
  8-15   shading maximum intensity   (unsigned, 00h..FFh)
  16-23  shading density attenuation (unsigned, 00h..FFh)
```

10401488h - PICA(0122h) - GPUREG_GAS_LIGHT_Z_COLOR (R/W)
```
  0-7    View shading effect in line-of-sight direction (unsigned, 00h..FFh)
  8      Gas color LUT input    (0=Gas density, 1=Light factor)
```
Configures gas light shading in the line-of-sight direction, and the input to the gas color LUT.

1040148Ch - PICA(0123h) - GPUREG_GAS_LUT_INDEX (W)
```
  0-15   Index... uh does that really have 16bit range? (W)
```
Index for writing GPUREG_GAS_LUT_DATAi.

10401490h - PICA(0124h) - GPUREG_GAS_LUT_DATA (R/W)
```
  0-31   LUT data
```
These registers behave as a FIFO queue. Each write to these registers writes the provided value to the gas look-up table, starting at the index selected with GPUREG_GAS_LUT_INDEX. Uh, how many registers are "these registers"?
Gas Look-Up Table (16 elements):
```
 First 8 elements:
  0-7    Red    (signed, -80h..+7Fh)
  8-15   Green  (signed, -80h..+7Fh)
  16-23  Blue   (signed, -80h..+7Fh)
 Last 8 elements:
  0-7    Red    (unsigned, 00h..FFh)
  8-15   Green  (unsigned, 00h..FFh)
  16-23  Blue   (unsigned, 00h..FFh)
```

10401494h - PICA(0125h) - GPUREG_undoc_10401494h (R/W=0000FFFFh)
```
  0-15   UNKNOWN (0..FFFFh) (R/W)
```

10401498h - PICA(0126h) - GPUREG_GAS_DELTAZ_DEPTH (R/W)
```
  0-23   Depth direction attenuation proportion (fixed0.16.8)
  24-25  Depth function (0..3, see below)
```
Configures the gas depth direction attenuation proportion, as well as the gas depth function.
```
 Gas depth function values:
  0h = Never
  1h = Always
  2h = Greater than/Greater than or equal            ;uh?
  3h = Less than/Less than or equal/Equal/Not equal  ;uh??
```

104014C0h - PICA(0130h) - GPUREG_FRAGOP_SHADOW (R/W)
```
  0-15   float1.5.10, Sum of penumbra scale and penumbra bias
  16-31  float1.5.10, Penumbra scale with reversed sign
```
Configures shadow properties.

104014FCh - PICA(013Fh) - GPUREG_undoc_104014FCh (R/W=0000000Fh)
```
  0-15   UNKNOWN (0..0Fh) (R/W)
```

3DS GPU Internal Registers - Texturing registers (Generic Textures)

10401200h - PICA(0080h) - GPUREG_TEXUNIT_CONFIG (R/W)
```
  0      Texture 0 enable      (0=Disable, 1=Enable)                    (R/W)
```

```
  1      Texture 1 enable      (0=Disable, 1=Enable)                    (R/W)
  2      Texture 2 enable      (0=Disable, 1=Enable)                    (R/W)
  3      UNKNOWN "0x0" (R/W)                                            (R/W)
  8-9    Texture 3 coordinates (0=Texture0, 1=Texture1, 2=Texture2)     (R/W)
  10     Texture 3 enable      (0=Disable, 1=Enable)                    (R/W)
  12     UNKNOWN "0x1" (R/W)                                            (R/W)
  13     Texture 2 coordinates (0=Texture2, 1=Texture1)                 (R/W)
  16     Clear texture cache   (0=No change, 1=Clear)                   (W)
  17-31  0x0, uh, what???                                               (??)
  24-25  UNKNOWN (R/W)                                                  (R/W)
```

10401204h - PICA(0081h) - GPUREG_TEXUNIT0_BORDER_COLOR (R/W)
10401244h - PICA(0091h) - GPUREG_TEXUNIT1_BORDER_COLOR (R/W)
10401264h - PICA(0099h) - GPUREG_TEXUNIT2_BORDER_COLOR (R/W)

```
  0-7    Border color Red    (unsigned, 00h..FFh)
  8-15   Border color Green  (unsigned, 00h..FFh)
  16-23  Border color Blue   (unsigned, 00h..FFh)
  24-31  Border color Alpha  (unsigned, 00h..FFh)
```

10401208h - PICA(0082h) - GPUREG_TEXUNIT0_DIM (R/W)
10401248h - PICA(0092h) - GPUREG_TEXUNIT1_DIM (R/W)
10401268h - PICA(009Ah) - GPUREG_TEXUNIT2_DIM (R/W)

```
  0-10   Texture dimension Height (unsigned, 0..7FFh)
  11-15  UNKNOWN (R/W)
  16-26  Texture dimension Width  (unsigned, 0..7FFh)
```

1040120Ch - PICA(0083h) - GPUREG_TEXUNIT0_PARAM (R/W)
1040124Ch - PICA(0093h) - GPUREG_TEXUNIT1_PARAM (R/W)
1040126Ch - PICA(009Bh) - GPUREG_TEXUNIT2_PARAM (R/W)

```
  0      UNKNOWN (R/W)
  1      Magnification filter   (0=Nearest, 1=Linear)
  2      Minification filter    (0=Nearest, 1=Linear)
  3      UNKNOWN (R/W)
  4-5    ETC1   (0=NotETC1, 2=ETC1, 1/3=?)  ;note: still 0 for ETC1A4
  6-7    UNKNOWN (R/W)
  8-10   Wrap T (0=ClampToEdge, 1=ClampToBorder, 2=Repeat, 3=MirroredRepeat)
  11     UNKNOWN (R/W)
  12-14  Wrap S (0=ClampToEdge, 1=ClampToBorder, 2=Repeat, 3=MirroredRepeat)
  15     UNKNOWN (R/W)
  16-17  UNKNOWN "0x0" (R/W)
  18-19  UNKNOWN (R/W)
  20     TEXUNIT0 only: Shadow (0=No, 1=Shadow)
  20     TEXUNIT1/2: UNKNOWN (R/W)
  21-23  UNKNOWN (R/W)
  24     Mipmap filter          (0=Nearest, 1=Linear)
  28-30  TEXUNIT0 only: Type   (see below)
  28-30  TEXUNIT1/2: Unused (0)
```

Type values:

```
  0=2D, 1=CubeMap, 2=Shadow2D, 3=Projection, 4=ShadowCube, 5=Disabled
```

10401210h - PICA(0084h) - GPUREG_TEXUNIT0_LOD (R/W)
10401250h - PICA(0094h) - GPUREG_TEXUNIT1_LOD (R/W)
10401270h - PICA(009Ch) - GPUREG_TEXUNIT2_LOD (R/W)

```
  0-12   fixed1.4.8, Bias
  13-15  UNKNOWN (R/W)
  16-19  Max Level of Detail (max LOD) (unsigned, 0..0Fh = ?)
  20-23  UNKNOWN (R/W)
  24-27  Min Level of Detail (min LOD) (unsigned, 0..0Fh = ?)
```

10401214h - PICA(0085h) - GPUREG_TEXUNIT0_ADDR1 (R/W)
10401218h - PICA(0086h) - GPUREG_TEXUNIT0_ADDR2 (R/W)

1040121Ch - PICA(0087h) - GPUREG_TEXUNIT0_ADDR3 (R/W)
10401220h - PICA(0088h) - GPUREG_TEXUNIT0_ADDR4 (R/W)
10401224h - PICA(0089h) - GPUREG_TEXUNIT0_ADDR5 (R/W)
10401228h - PICA(008Ah) - GPUREG_TEXUNIT0_ADDR6 (R/W)
10401254h - PICA(0095h) - GPUREG_TEXUNIT1_ADDR (R/W)
10401274h - PICA(009Dh) - GPUREG_TEXUNIT2_ADDR (R/W)
First ADDR register, PICA(0085h,0095h,009Dh):

```
  0-27   Texture physical memory address, in 8-byte units (full 28bits)
```

Subsequent ADDR registers, PICA(0086h..008Ah):

```
  0-21   Texture physical memory address, in 8-byte units (lower 22bits)
         (upper 6bits are reused from first ADDR register)
```

Individual texels in a texture are laid out in memory as a Z-order curve. Mipmap data is stored directly following the main texture data.
If the texture is a cube:

```
  Register Description
  ADDR1    Positive X
  ADDR2    Negative X
  ADDR3    Positive Y
  ADDR4    Negative Y
  ADDR5    Positive Z
  ADDR6    Negative Z
```

Otherwise, ADDR(1) points to a 2D texture, and the rest are empty.

1040122Ch - PICA(008Bh) - GPUREG_TEXUNIT0_SHADOW (R/W)

```
  0      Perspective (0=Perspective, 1=Not perspective)
  1-23   fixed0.0.24, Z bias (upper 23 bits) (uh, how can that have 24bit?)
  24-31  UNKNOWN (R/W)
```

"Texture unit's shadow texture properties."

10401230h - PICA(008Ch) - GPUREG_undoc_10401230h (R/W=FFFF00FFh)

```
  0-7    UNKNOWN (0..FFh) (R/W)
  16-31  UNKNOWN (0..FFFFh) (R/W)
```

10401234h - PICA(008Dh) - GPUREG_undoc_10401234h (R/W=000000FFh)

```
  0-7    UNKNOWN (0..FFh) (R/W)
```

10401238h - PICA(008Eh) - GPUREG_TEXUNIT0_TYPE (R/W)
10401258h - PICA(0096h) - GPUREG_TEXUNIT1_TYPE (R/W)
10401278h - PICA(009Eh) - GPUREG_TEXUNIT2_TYPE (R/W)

```
  0-3    Texture Data Format
```

Format values:

```
  00h = RGBA8888        GL_RGBA            GL_UNSIGNED_BYTE
  01h = RGB888          GL_RGB             GL_UNSIGNED_BYTE
  02h = RGBA5551        GL_RGBA            GL_UNSIGNED_SHORT_5_5_5_1
  03h = RGB565          GL_RGB             GL_UNSIGNED_SHORT_5_6_5
  04h = RGBA4444        GL_RGBA            GL_UNSIGNED_SHORT_4_4_4_4
  05h = IA8             GL_LUMINANCE_ALPHA GL_UNSIGNED_BYTE
  06h = HILO8
  07h = I8              GL_LUMINANCE       GL_UNSIGNED_BYTE
  08h = A8              GL_ALPHA           GL_UNSIGNED_BYTE
  09h = IA44            GL_LUMINANCE_ALPHA GL_UNSIGNED_BYTE_4_4_EXT
  0Ah = I4
  0Bh = A4              GL_ALPHA           GL_UNSIGNED_NIBBLE_EXT
  0Ch = ETC1            GL_ETC1_RGB8_OES
  0Dh = ETC1A4
```

3DS GPU Internal Registers - Texturing registers (Procedural Texture)

These registers are for "Procedural Texture", whatever that means. Probably something for automatically generating or animating plasma/fire/water textures.
As far as known, these are controlled solely by the parameters in the registers below (without actually using/supporting custom program code on the shader units or so).
See GPUREG_TEXUNIT_CONFIG.bit8,9,10 for enable flag... and for selecting coordinates from texture unit 0, 1, or 2. Unknown what those coordinates are used for... as additional incoming PROCTEX parameters, or maybe the PROCTEX unit does output coordinates to the selected texture unit?

104012A0h - PICA(00A8h) - GPUREG_TEXUNIT3_PROCTEX0 (R/W)
```
  0-2    U-direction clamp       ;\clamp, see below
  3-5    V-direction clamp       ;/
  6-9    RGB mapping function    ;\function, see below
  10-13  Alpha mapping function  ;/
  14     Handle alpha separately   (0=Don't separate, 1=Separate)
  15     Noise enable              (0=Disable, 1=Enable)
  16-17  U-direction shift         (0=None, 1=Odd, 2=Even, 3=?)
  18-19  V-direction shift         (0=None, 1=Odd, 2=Even, 3=?)
  20-27  Texture bias (lower 8bit of float1.5.10) (upper 8bit are in PROCTEX4)
```
Clamp values:
```
  00h = Clamp to zero
  01h = Clamp to edge
  02h = Symmetrical repeat
  03h = Mirrored repeat
  04h = Pulse
```
Mapping function values:
```
  00h = U
  01h = U^2
  02h = V
  03h = V^2
  04h = (U + V) / 2
  05h = (U^2 + V^2) / 2
  06h = sqrt(U^2 + V^2)
  07h = Minimum
  08h = Maximum
  09h = Rmax
```

104012A4h - PICA(00A9h) - GPUREG_TEXUNIT3_PROCTEX1 (R/W)
```
  0-15   fixed1.3.12, U-direction noise amplitude
  16-31  float1.5.10, U-direction noise phase
```
Configures the procedural texture unit's U-direction noise amplitude/phase.

104012A8h - PICA(00AAh) - GPUREG_TEXUNIT3_PROCTEX2 (R/W)
```
  0-15   fixed1.3.12, V-direction noise amplitude
  16-31  float1.5.10, V-direction noise phase
```
Configures the procedural texture unit's V-direction noise amplitude/phase.

104012ACh - PICA(00ABh) - GPUREG_TEXUNIT3_PROCTEX3 (R/W)
```
  0-15   float1.5.10, U-direction noise frequency
  16-31  float1.5.10, V-direction noise frequency
```
Configure the procedural texture unit's U-direction and V-direction noise frequency.

104012B0h - PICA(00ACh) - GPUREG_TEXUNIT3_PROCTEX4 (R/W)
```
  0-2    Minification filter (see below)
  3-6    Min Level of Detail (Min LOD) (usually 0)
  7-10   Max Level of Detail (Max LOD) (usually 6)
  11-18  Texture width (unsigned, 00h..FFh)
  19-26  Texture bias (upper 8bit of float1.5.10) (lower 8bit are in PROCTEX0)
```
Minification filter values:
```
  00h = Nearest
```

```
  01h = Linear
  02h = Nearest, Mipmap Nearest
  03h = Linear, Mipmap Nearest
  04h = Nearest, Mipmap Linear
  05h = Linear, Mipmap Linear
```

104012B4h - PICA(00ADh) - GPUREG_TEXUNIT3_PROCTEX5_LOW (R/W)
104012B8h - PICA(00AEh) - GPUREG_TEXUNIT3_PROCTEX5_HIGH (R/W)

```
  0-7    Mipmap level 0 base level (usually 00h) (Texture offset)
  8-15   Mipmap level 1 offset     (usually 80h)
  16-23  Mipmap level 2 offset     (usually C0h)
  24-31  Mipmap level 3 offset     (usually E0h)
  32-39  Mipmap level 4 offset     (usually F0h)
  40-47  Mipmap level 5 offset     (usually F8h)
  48-55  Mipmap level 6 offset     (usually FCh)
  56-63  Mipmap level 7 offset     (usually FEh)
```

Usually set to LOW=E0C08000h, HIGH=FEFCF8F0h.
"Sets the procedural texture unit's offset."

104012BCh - PICA(00AFh) - GPUREG_PROCTEX_LUT_INDEX (R/W)

```
  0-7    Index           (0..255 or 0..127, depending on selected table)
  8-11   Reference table (0=Noise, 1=?, 2=RGB, 3=Alpha, 4=Color, 5=ColorDiff)
```

Selects look-up table and index for writing via GPUREG_PROCTEX_LUT_DATA(i).

104012C0h+i*4 - PICA(00B0h+i) - GPUREG_PROCTEX_LUT_DATA(0..7) (R/W)

```
  0-31   LUT data
```

These registers behave as a FIFO queue. Each write to these registers writes the provided value to the table selected with GPUREG_PROCTEX_LUT, starting at the selected index.
Noise Table (128 elements):

```
  0-11   fixed0.0.12, Value
  12-23  fixed0.0.12 with two's complement
           ( [0.5,1.0) mapped to [-1.0,0) ), Difference from next element
```

RGB Mapping Function Table (128 elements):

```
  0-11   fixed0.0.12, Value
  12-23  fixed0.0.12 with two's complement, Difference from next element
```

Alpha Mapping Function Table (128 elements):

```
  0-11   fixed0.0.12, Value
  12-23  fixed0.0.12 with two's complement, Difference from next element
```

Color Table (256 elements):

```
  0-7    Red           (unsigned, 00h..FFh)
  8-15   Green         (unsigned, 00h..FFh)
  16-23  Blue          (unsigned, 00h..FFh)
  24-31  Alpha         (unsigned, 00h..FFh)
```

Color Difference Table (256 elements):

```
  0-7    Half of red   (signed, -80h..+7Fh) ;\
  8-15   Half of green (signed, -80h..+7Fh) ; difference between current
  16-23  Half of blue  (signed, -80h..+7Fh) ; and next color table elements
  24-31  Half of alpha (signed, -80h..+7Fh) ;/
```

3DS GPU Internal Registers - Texturing registers (Environment)

Below are "texture env(ironment?)" registers for "texture combiner's". Unknown what that means, probably something for merging output from texture units 0,1,2. But unknown why there are six TexEnv's.

10401300h - PICA(00C0h) - GPUREG_TEXENV0_SOURCE (R/W)
10401320h - PICA(00C8h) - GPUREG_TEXENV1_SOURCE (R/W)
10401340h - PICA(00D0h) - GPUREG_TEXENV2_SOURCE (R/W)
10401360h - PICA(00D8h) - GPUREG_TEXENV3_SOURCE (R/W)

104013C0h - PICA(00F0h) - GPUREG_TEXENV4_SOURCE (R/W)
104013E0h - PICA(00F8h) - GPUREG_TEXENV5_SOURCE (R/W)

```
  0-3    RGB source 0
  4-7    RGB source 1
  8-11   RGB source 2
  16-19  Alpha source 0
  20-23  Alpha source 1
  24-27  Alpha source 2
```

Configures a texture combiner's sources.
Source values:

```
  00h = Primary color
  01h = Fragment primary color
  02h = Fragment secondary color
  03h = Texture 0
  04h = Texture 1
  05h = Texture 2
  06h = Texture 3
  0xh = ?
  0Dh = Previous buffer
  0Eh = Constant (from GPUREG_TEXENVi_COLOR)
  0Fh = Previous
```

10401304h - PICA(00C1h) - GPUREG_TEXENV0_OPERAND (R/W)
10401324h - PICA(00C9h) - GPUREG_TEXENV1_OPERAND (R/W)
10401344h - PICA(00D1h) - GPUREG_TEXENV2_OPERAND (R/W)
10401364h - PICA(00D9h) - GPUREG_TEXENV3_OPERAND (R/W)
104013C4h - PICA(00F1h) - GPUREG_TEXENV4_OPERAND (R/W)
104013E4h - PICA(00F9h) - GPUREG_TEXENV5_OPERAND (R/W)
"Configures a texture combiner's operands."
uh, R/W mask is only 00111333h...?

```
  0-3    RGB operand 0    (0..0Dh, see below)     ;\uh, but R/W mask suggest
  4-7    RGB operand 1    (0..0Dh, see below)     ; range 0..3 only
  8-11   RGB operand 2    (0..0Dh, see below)     ;/
  12-14  Alpha operand 0  (0..7, see below)       ;\uh, but R/W mask suggest
  16-18  Alpha operand 1  (0..7, see below)       ; range 0..1 only
  20-22  Alpha operand 2  (0..7, see below)       ;/
```

RGB operand values:

```
  00h = Source color
  01h = One minus source color
  02h = Source alpha
  03h = One minus source alpha
  04h = Source red                  ;\
  05h = One minus source red        ;
  08h = Source green                ; uh, really? R/W mask is only 2bit
  09h = One minus source green      ;
  0Ch = Source blue                 ;
  0Dh = One minus source blue       ;/
```

Alpha operand values:

```
  0h  = Source alpha
  1h  = One minus source alpha
  2h  = Source red                  ;\
  3h  = One minus source red        ;
  4h  = Source green                ; uh, really? R/W mask is only 1bit
  5h  = One minus source green      ;
  6h  = Source blue                 ;
  7h  = One minus source blue       ;/
```

10401308h - PICA(00C2h) - GPUREG_TEXENV0_COMBINER (R/W)
10401328h - PICA(00CAh) - GPUREG_TEXENV1_COMBINER (R/W)
10401348h - PICA(00D2h) - GPUREG_TEXENV2_COMBINER (R/W)
10401368h - PICA(00DAh) - GPUREG_TEXENV3_COMBINER (R/W)

104013C8h - PICA(00F2h) - GPUREG_TEXENV4_COMBINER (R/W)
104013E8h - PICA(00FAh) - GPUREG_TEXENV5_COMBINER (R/W)

```
  0-3    RGB combine mode   (0..9, see below)
  16-19  Alpha combine mode (0..9, see below)
```

Combine mode values:

```
  0h = Replace
  1h = Modulate
  2h = Add
  3h = Add signed
  4h = Interpolate
  5h = Subtract
  6h = Dot3 RGB
  7h = Dot3 RGBA
  8h = Multiply then add
  9h = Add then multiply
  Ah..Fh = ?
```

1040130Ch - PICA(00C3h) - GPUREG_TEXENV0_COLOR (R/W)
1040132Ch - PICA(00CBh) - GPUREG_TEXENV1_COLOR (R/W)
1040134Ch - PICA(00D3h) - GPUREG_TEXENV2_COLOR (R/W)
1040136Ch - PICA(00DBh) - GPUREG_TEXENV3_COLOR (R/W)
104013CCh - PICA(00F3h) - GPUREG_TEXENV4_COLOR (R/W)
104013ECh - PICA(00FBh) - GPUREG_TEXENV5_COLOR (R/W)

```
  0-7    Constant color Red    (unsigned, 00h..FFh)
  8-15   Constant color Green  (unsigned, 00h..FFh)
  16-23  Constant color Blue   (unsigned, 00h..FFh)
  24-31  Constant color Alpha  (unsigned, 00h..FFh)
```

10401310h - PICA(00C4h) - GPUREG_TEXENV0_SCALE (R/W)
10401330h - PICA(00CCh) - GPUREG_TEXENV1_SCALE (R/W)
10401350h - PICA(00D4h) - GPUREG_TEXENV2_SCALE (R/W)
10401370h - PICA(00DCh) - GPUREG_TEXENV3_SCALE (R/W)
104013D0h - PICA(00F4h) - GPUREG_TEXENV4_SCALE (R/W)
104013F0h - PICA(00FCh) - GPUREG_TEXENV5_SCALE (R/W)

```
  0-1    RGB scale      (0=1x, 1=2x, 2=4x, 3=?)
  16-17  Alpha scale    (0=1x, 1=2x, 2=4x, 3=?)
```

1040131Ch - PICA(00C7h) - GPUREG_undoc_1040131Ch (R/W=00000007h)

```
  0-2    UNKNOWN (0..7) (R/W)
```

10401380h - PICA(00E0h) - GPUREG_TEXENV_UPDATE_BUFFER (R/W)

```
  0-2    Fog mode                     (0=Disabled, 5=Fog, 7=Gas, other=?)
  3      Shading density source       (0=Plain density, 1=Depth density)
  8      TexEnv 1 RGB buffer input    (0=Previous Buffer, 1=Previous)
  9      TexEnv 2 RGB buffer input    (0=Previous Buffer, 1=Previous)
  10     TexEnv 3 RGB buffer input    (0=Previous Buffer, 1=Previous)
  11     TexEnv 4 RGB buffer input    (0=Previous Buffer, 1=Previous)
  12     TexEnv 1 Alpha buffer input  (0=Previous Buffer, 1=Previous)
  13     TexEnv 2 Alpha buffer input  (0=Previous Buffer, 1=Previous)
  14     TexEnv 3 Alpha buffer input  (0=Previous Buffer, 1=Previous)
  15     TexEnv 4 Alpha buffer input  (0=Previous Buffer, 1=Previous)
  16     Z flip                       (0=Don't flip,1=Flip)
  24-25  UNKNOWN "0x0" (R/W)
```

This register is shared between the gas/fog mode configuration and texture combiner buffer inputs. Texture combiner buffer inputs are typically written with a mask of 0x2, and the gas/fog mode configuration is typically written with a mask of 0x5.

10401384h - PICA(00E1h) - GPUREG_FOG_COLOR (R/W)

```
  0-7    Fog color Red    (unsigned, 00h..FFh)
  8-15   Fog color Green  (unsigned, 00h..FFh)
```

```
16-23  Fog color Blue   (unsigned, 00h..FFh)
```

10401388h - PICA(00E2h) - GPUREG_undoc_10401388h (R/W=0000FFFFh)
1040138Ch - PICA(00E3h) - GPUREG_undoc_1040138Ch (R/W=0000FFFFh)
```
0-15   UNKNOWN (0..FFFFh) (R/W)
```

10401390h - PICA(00E4h) - GPUREG_GAS_ATTENUATION (R/W)
```
0-15   float1.5.10, Gas density attenuation
```

10401394h - PICA(00E5h) - GPUREG_GAS_ACCMAX (R/W)
```
0-15   float1.5.10, Gas maximum density accumulation
```

10401398h - PICA(00E6h) - GPUREG_FOG_LUT_INDEX (R/W)
```
0-6    Index for FOG_LUT_DATA (0..7Fh)
```
The index is incremented on FOG_LUT_DATA writes (but not incremented on reads).

104013A0h+i*4 - PICA(00E8h+i) - GPUREG_FOG_LUT_DATA(0..7) (R/W)
```
0-12   fixed1.1.11, Difference from next entry
13-23  fixed0.0.11, Value
```
These registers behave as a FIFO queue. Each write to these registers writes the provided value to the fog look-up table, starting at the index selected with GPUREG_FOG_LUT_INDEX.

104013F4h - PICA(00FDh) - GPUREG_TEXENV_BUFFER_COLOR (R/W)
```
0-7    Texture combiner buffer color Red    (unsigned, 00h..FFh)
8-15   Texture combiner buffer color Green  (unsigned, 00h..FFh)
16-23  Texture combiner buffer color Blue   (unsigned, 00h..FFh)
24-31  Texture combiner buffer color Alpha  (unsigned, 00h..FFh)
```

3DS GPU Internal Registers - Fragment Lighting registers

10401500h+i*40h - PICA(0140h+10h*(0..7)) - GPUREG_LIGHTi_SPECULAR0 (R/W)
10401504h+i*40h - PICA(0141h+10h*(0..7)) - GPUREG_LIGHTi_SPECULAR1 (R/W)
10401508h+i*40h - PICA(0142h+10h*(0..7)) - GPUREG_LIGHTi_DIFFUSE (R/W)
1040150Ch+i*40h - PICA(0143h+10h*(0..7)) - GPUREG_LIGHTi_AMBIENT (R/W)
10401700h - PICA(01C0h) - GPUREG_LIGHTING_AMBIENT (R/W)
```
0-7    unsigned, Blue
8-9    UNKNOWN (maybe Blue MSBs? if so, is it really unsigned?) (R/W)
10-17  unsigned, Green
18-19  UNKNOWN (R/W)
20-27  unsigned, Red
28-29  UNKNOWN (R/W)
```
These registers contain the colors of the corresponding lights, usually set to following values:
```
GPUREG_LIGHTi_SPECULAR0 = material_specular0*lightX_specular0
GPUREG_LIGHTi_SPECULAR1 = material_specular1*lightX_specular1
GPUREG_LIGHTi_DIFFUSE   = material_diffuse*lightX_diffuse
GPUREG_LIGHTi_AMBIENT   = material_ambient*lightX_ambient
GPUREG_LIGHTING_AMBIENT = material_ambient*scene_ambient + material_emission
```

10401510h+i*40h - PICA(0144h+10h*(0..7)) - GPUREG_LIGHTi_VECTOR_LOW (R/W)
10401514h+i*40h - PICA(0145h+10h*(0..7)) - GPUREG_LIGHTi_VECTOR_HIGH (R/W)
```
0-15   float1.5.10, X coordinate
16-31  float1.5.10, Y coordinate
32-47  float1.5.10, Z coordinate
```
Light position (for a positional light) or the light direction vector (for a directional light) of the corresponding light.
Uh, does the direction vector need to be normalized? And how does it differ from SPOTDIR direction vector?

10401518h+i*40h - PICA(0146h+10h*(0..7)) - GPUREG_LIGHTi_SPOTDIR_LOW (R/W)
1040151Ch+i*40h - PICA(0147h+10h*(0..7)) - GPUREG_LIGHTi_SPOTDIR_HIGH (R/W)
```
  0-12   fixed1.1.11, X coordinate (negated)
  16-28  fixed1.1.11, Y coordinate (negated)
  32-44  fixed1.1.11, Z coordinate (negated)
```
Spot direction (unitary) vector of the corresponding light.

10401524h+i*40h - PICA(0149h+10h*(0..7)) - GPUREG_LIGHTi_CONFIG (R/W)
```
  0      Light type             (0=Positional light, 1=Directional light)
  1      Two side diffuse       (0=One side, 1=Both sides)
  2      Use geometric factor 0 (0=Don't use, 1=Use)
  3      Use geometric factor 1 (0=Don't use, 1=Use)
```
Configures a light's properties.

10401528h+i*40h - PICA(014Ah+10h*(0..7)) - GPUREG_LIGHTi_ATTENUATION_BIAS
1040152Ch+i*40h - PICA(014Bh+10h*(0..7)) - GPUREG_LIGHTi_ATTENUATION_SCALE
```
  0-19   float1.7.12, Distance attenuation value    (R/W)
```
For the odd 20bit floating point format, one could do:
```
  (float24)/10h                ;convert float1.7.16 to float1.7.12
```
Distance attenuation BIAS and SCALE values of the corresponding light.
The attenuation factor is DA(clip(BIAS + SCALE*distance, 0.0, 1.0)).

1040123Ch - PICA(008Fh) - GPUREG_LIGHTING_ENABLE (R/W)
```
  0      Enable Lighting  (0=Disable, 1=Enable)
```
Uh, that is in texture register area... is that texture specific?

10401718h - PICA(01C6h) - GPUREG_LIGHTING_DISABLE (R/W)
```
  0      Disable (0=Enable, 1=Disable)
```

10401708h - PICA(01C2h) - GPUREG_LIGHTING_NUM_LIGHTS (R/W)
```
  0-2    Number of active lights - 1    (0..7 = 1..8 Lights)
```
To use 0 lights, one seems to have to disable lighting... unknown if one can keep GPUREG_LIGHTING_AMBIENT enabled in that state.

1040170Ch - PICA(01C3h) - GPUREG_LIGHTING_CONFIG0 (R/W)
```
  0      Shadow factor enable  (0=Disable, 1=Enable)
           (usually accompanied by bit 16, 17, or 18)
  2-3    Fresnel selector (0=None, 1=Primary, 2=Secondary, 3=Both alpha's)
  4-7    Light environment configuration (see below)
  8-11   UNKNOWN "0x4" (R/W)
  16     Apply shadow attenuation to primary color   (0=Don't apply, 1=Apply)
  17     Apply shadow attenuation to secondary color (0=Don't apply, 1=Apply)
  18     Invert shadow attenuation                 (0=Don't invert, 1=Invert)
  19     Apply shadow attenuation to alpha component (0=Don't apply, 1=Apply)
  20-21  UNKNOWN (R/W)
  22-23  Bump map texture unit
  24-25  Shadow map texture unit
  26     UNKNOWN (R/W)
  27     Clamp highlights         (0=Disable, 1=Enable)
  28-29  Bump mode (0=Not used, 1=Use as bump map, 2=Use as tangent map)
  30     Recalculate bump vectors (0=Enable, 1=Disable)
           (usually set to 1 when bump mode is not 0)
  31     UNKNOWN "0x1" (R/W)
```
Fresnel selector values:
```
  0h = None
  1h = Primary alpha
  2h = Secondary alpha
  3h = Primary and secondary alpha
```

Light environment configuration values:

```
  Value Description       Available LUTs
  00h = Configuration 0   D0, RR, SP, DA
  01h = Configuration 1   FR, RR, SP, DA
  02h = Configuration 2   D0, D1, RR, DA
  03h = Configuration 3   D0, D1, FR, DA
  04h = Configuration 4   All except for FR
  05h = Configuration 5   All except for D1
  06h = Configuration 6   All except for RB and RG
  08h = Configuration 7   All
```

The light environment configuration controls which LUTs are available for use. If a LUT is not available in the selected configuration, its value will always read a constant 1.0 regardless of the enable state in GPUREG_LIGHTING_CONFIG1. If RR is enabled but not RG or RB, the output of RR is used for the three components; Red, Green and Blue.

10401710h - PICA(01C4h) - GPUREG_LIGHTING_CONFIG1 (R/W)

```
  0-7   Fragment light source 0-7 shadows             (0=Enable, 1=Disable)
  8-15  Fragment light source 0-7 spot light          (0=Enable, 1=Disable)
  16    Term 0 distribution component D0 LUT          (0=Enable, 1=Disable)
  17    Term 1 distribution component D1 LUT          (0=Enable, 1=Disable)
  18    0x1                                           (1)
  19    Fresnel FR LUT disabled                       (0=Enable, 1=Disable)
  20    Term 1 reflection component RB LUT            (0=Enable, 1=Disable)
  21    Term 1 reflection component RG LUT            (0=Enable, 1=Disable)
  22    Term 1 reflection component RR LUT            (0=Enable, 1=Disable)
  24-31 Fragment light source 0-7 distance attenuation (0=Enable, 1=Disable)
```

Allows to disable various aspects of the light environment.

10401714h - PICA(01C5h) - GPUREG_LIGHTING_LUT_INDEX (R/W)

```
  0-7   Starting index       (00h..FFh)    ;\for reading/writing via
  8-12  Look-up table number (see below)   ;/GPUREG_LIGHTING_LUT_DATA(i)
```

Lookup table numbers:

```
  00h     = D0      ;\Distribution, whatever that is?
  01h     = D1      ;/
  03h     = FR      ;-Fresnel, whatever that is?
  04h     = RB      ;\
  05h     = RG      ; Reflection with separate blue/green/red tables?
  06h     = RR      ;/
  08h-0Fh = SP0-7   ;-Spotlight?             ;\maybe for light 0..7 ?
  10h-17h = DA0-7   ;-Distance attenuation?  ;/
```

Note: The index in bit0-7 is auto-incremented when writing (or reading) LUT_DATA registers; there is no carry-out to table number in bit8-12.

10401720h+i*4 - PICA(01C8h+i) - GPUREG_LIGHTING_LUT_DATA(0..7) (R/W)

```
  0-11   fixed0.0.12, Entry value
  12-23  fixed1.0.11, Absolute value of the difference between the next
           entry and this entry, used to implement linear interpolation
           uh, how/why is that a ABSOLUTE value WITH SIGN bit?
```

A LUT contains data for the input domain [-1.0, 1.0], which is indexed using a signed 8bit number [-128, 127]. Therefore a LUT contains 256 entries. The index of a value is (int)(x/127.0f) & 0xFF.
DA: The input domain is [0.0, 1.0], and the index is an unsigned 8bit number [0, 255] instead.

10401740h - PICA(01D0h) - GPUREG_LIGHTING_LUTINPUT_ABS (R/W)

```
  0      UNKNOWN (R/W)
  1      abs() flag for the input of D0 (0=Enable, 1=Disable)
  4      UNKNOWN (R/W)
  5      abs() flag for the input of D1 (0=Enable, 1=Disable)
  8      UNKNOWN (R/W)
  9      abs() flag for the input of SP (0=Enable, 1=Disable)
  12     UNKNOWN (R/W)
  13     abs() flag for the input of FR (0=Enable, 1=Disable)
```

```
  16     UNKNOWN (R/W)
  17     abs() flag for the input of RB (0=Enable, 1=Disable)
  20     UNKNOWN (R/W)
  21     abs() flag for the input of RG (0=Enable, 1=Disable)
  24     UNKNOWN (R/W)
  25     abs() flag for the input of RR (0=Enable, 1=Disable)
```
Controls whether the absolute value of the input is taken before using a LUT.

10401744h - PICA(01D1h) - GPUREG_LIGHTING_LUTINPUT_SELECT (R/W)
```
  0-2    Input selector for D0  ;\Input selector values:
  4-6    Input selector for D1  ;  0h = N . H   ;uh, dot symbol?
  8-10   Input selector for SP  ;  1h = V . H   ; ... maybe multiply?
  12-14  Input selector for FR  ;  2h = N . V   ;uh, who is N,H,L,V,P
  16-18  Input selector for RB  ;  3h = L . N   ; and greek symbol?
  20-22  Input selector for RG  ;  4h = -L . P (aka Spotlight aka SP)
  24-26  Input selector for RR  ;/ 5h = cos <greek symbol> (aka CP)
```
Selects the input from LUTs.

10401748h - PICA(01D2h) - GPUREG_LIGHTING_LUTINPUT_SCALE (R/W)
```
  0-2    Scaler selector for D0    ;\Scaler selector values:
  4-6    Scaler selector for D1    ;  0h = 1x
  8-10   Scaler selector for SP    ;  1h = 2x
  12-14  Scaler selector for FR    ;  2h = 4x
  16-18  Scaler selector for RB    ;  3h = 8x
  20-22  Scaler selector for RG    ;  6h = 0.25x
  24-26  Scaler selector for RR    ;/ 7h = 0.5x
  28-30  UNKNOWN (R/W)
```
Controls the scaling that is applied to the output of a LUT.

1040174Ch - PICA(01D3h) - GPUREG_undoc_1040174Ch (R/W=00000001h)
```
  0      UNKNOWN (0..1) (R/W)
```

10401750h - PICA(01D4h) - GPUREG_undoc_10401750h (R/W=0FFFFF03h)
```
  0-1    UNKNOWN (0..3) (R/W)
  8-31   UNKNOWN (0..FFFFFFh) (R/W)
```

10401754h - PICA(01D5h) - GPUREG_undoc_10401754h (R/W=1FFF1FFFh)
10401758h - PICA(01D6h) - GPUREG_undoc_10401758h (R/W=1FFF1FFFh)
```
  0-12   UNKNOWN (0..1FFFh) (R/W)
  16-28  UNKNOWN (0..1FFFh) (R/W)
```

1040175Ch - PICA(01D7h) - GPUREG_undoc_1040175Ch (R/W=000FFFFFh)
10401760h - PICA(01D8h) - GPUREG_undoc_10401760h (R/W=000FFFFFh)
```
  0-19   UNKNOWN (0..FFFFFh) (R/W)
```

10401764h - PICA(01D9h) - GPUREG_LIGHTING_LIGHT_PERMUTATION (R/W)
```
  0-2    ID of the 1st enabled light (0..7)
  4-6    ID of the 2nd enabled light (0..7)
  8-10   ID of the 3rd enabled light (0..7)
  12-14  ID of the 4th enabled light (0..7)
  16-18  ID of the 5th enabled light (0..7)
  20-22  ID of the 6th enabled light (0..7)
  24-26  ID of the 7th enabled light (0..7)
  28-30  ID of the 8th enabled light (0..7)
```
Sets the IDs of enabled light sources (1st..8th light does refer to the number of lights in GPUREG_LIGHTING_NUM_LIGHTS).
Uh, LIGHT_PERMUTATION is a scary homebrew slang for LIGHT_IDs?

3DS GPU Internal Registers - Unknown/Unused/Undocumented Registers

Unused Registers that Mirror to GPUREG_RESTART_PRIMITIVE (R/W)

```
104011C0h 40h  ;PICA(0070h..007Fh) mirrors to GPUREG_RESTART_PRIMITIVE
104013F8h 08h  ;PICA(00FEh..00FFh) mirrors to GPUREG_RESTART_PRIMITIVE
1040149Ch 24h  ;PICA(0127h..012Fh) mirrors to GPUREG_RESTART_PRIMITIVE
10401780h 80h  ;PICA(01E0h..01FFh) mirrors to GPUREG_RESTART_PRIMITIVE
10401900h 08h  ;PICA(0240h..0241h) mirrors to GPUREG_RESTART_PRIMITIVE
10401918h 10h  ;PICA(0246h..0249h) mirrors to GPUREG_RESTART_PRIMITIVE
1040192Ch 14h  ;PICA(024Bh..024Fh) mirrors to GPUREG_RESTART_PRIMITIVE
10401960h 18h  ;PICA(0258h..025Dh) mirrors to GPUREG_RESTART_PRIMITIVE
10401980h 80h  ;PICA(0260h..027Fh) mirrors to GPUREG_RESTART_PRIMITIVE
10401D00h 300h ;PICA(0340h..03FFh) mirrors to GPUREG_RESTART_PRIMITIVE
```

The mirrors (and GPUREG_RESTART_PRIMITIVE itself) are R/W, however, writing doesn't always work for some reason, or the write is applied only after issuing another three writes or so?

Unused or Write-Only Registers that return Garbage/Mirrors when reading

104018BCh..104018DCh and 104018F0h..104018FCh do return mirrors of multiple other registers ORed together.

```
104018BCh = 1040183Ch,1040189Ch,104018B4h           ;ATTR_BUF_DRAWELEMENTS
104018C0h = 10401840h,104018E0h,10401880h           ;unused (or write-only?)
104018C4h = 10401844h,104018E4h,10401884h           ;VTX_FUNC
104018C8h = 10401848h,104018E8h,10401888h           ;FIXEDATTRIB_INDEX
104018CCh = 1040184Ch,104018ECh,1040188Ch,1040189Ch ;FIXEDATTRIB_DATA0
104018D0h = 10401850h,104018E0h,10401890h           ;FIXEDATTRIB_DATA1
104018D4h = 10401854h,104018E4h,10401894h           ;FIXEDATTRIB_DATA2
104018D8h = 10401858h,104018E8h,10401898h           ;unused (or write-only?)
104018DCh = 1040185Ch,104018ECh                     ;unused (or write-only?)
104018F0h = 10401870h,104018E0h                     ;GPUREG_CMDBUF_JUMP0
104018F4h = 10401874h,104018E4h,104018B4h           ;GPUREG_CMDBUF_JUMP1
104018F8h = 10401878h,104018E8h                     ;unused (or write-only?)
104018FCh = 1040187Ch,104018ECh,104018B4h,1040189Ch ;unused (or write-only?)
```

10401940h, 10401958h, 1040195Ch seem to mirror to single registers without ORing multiple values.

```
10401940h = 10401944h (GPUREG_VSH_OUTMAP_TOTAL2)    ;unused (or write-only?)
10401958h = 10401950h (GPUREG_GSH_MISC1)            ;unused (or write-only?)
1040195Ch = 1040194Ch (GPUREG_GEOSTAGE_CONFIG2)     ;unused (or write-only?)
```

Unused or Write-Only Registers that return Zero when reading

```
1040118Ch = zero                                    ;EARLYDEPTH_CLEAR
10401440h = zero                                    ;RENDERBUFFER_INVALIDATE
10401444h = zero                                    ;RENDERBUFFER_FLUSH
1040148Ch = zero                                    ;GAS_LUT_INDEX
104018B8h = zero                                    ;ATTR_BUF_DRAWARRAYS
10401A14h/10401AD4h/10401B94h/10401C54h     = zero  ;unused (or write-only?)
10401A18h/10401AD8h/10401B98h/10401C58h     = zero  ;unused (or write-only?)
10401A1Ch/10401ADCh/10401B9Ch/10401C5Ch     = zero  ;unused (or write-only?)
10401A20h/10401AE0h/10401BA0h/10401C60h     = zero  ;unused (or write-only?)
10401A38h/10401AF8h/10401BB8h/10401C78h     = zero  ;unused (or write-only?)
10401A3Ch/10401AFCh/10401BBCh/10401C7Ch     = zero  ;CODETRANSFER_END
10401A40h/10401B00h/10401BC0h/10401C80h     = zero  ;FLOATUNIFORM_INDEX
10401A44h/10401B04h/10401BC4h/10401C84h+i*4 = zero  ;FLOATUNIFORM_DATA0-7
10401A64h/10401B24h/10401BE4h/10401CA4h     = zero  ;unused (or write-only?)
10401A68h/10401B28h/10401BE8h/10401CA8h     = zero  ;unused (or write-only?)
10401A6Ch/10401B2Ch/10401BECh/10401CACh     = zero  ;CODETRANSFER_INDEX
10401A70h/10401B30h/10401BF0h/10401CB0h+i*4 = zero  ;CODETRANSFER_DATA0-7
10401A90h/10401B50h/10401B10h/10401CD0h     = zero  ;unused (or write-only?)
10401A94h/10401B54h/10401C14h/10401CD4h     = zero  ;OPDESCS_INDEX
10401A98h/10401B58h/10401C18h/10401CD8h+i*4 = zero  ;OPDESCS_DATA0..7
10401AB8h/10401B78h/10401C38h/10401CF8h     = zero  ;unused (or write-only?)
10401ABCh/10401B7Ch/10401C3Ch/10401CFCh     = zero  ;unused (or write-only?)
```

Unused Registers that return Zero when reading
These registers aren't R/W, and they seem to be always zero (though some might be undocumented read-only or write-only registers).

```
104010D8h 28h  ;PICA(N/A)          unused (0)           ;-Finalize
1040115Ch 4    ;PICA(0057h)        unused (0)
10401174h 0Ch  ;PICA(005Dh..005Fh) unused (0)
10401240h 4    ;PICA(0090h)        unused (0)   ;XXX used, write-only?
1040125Ch 4    ;PICA(0097h)        unused (0)   ;XXX used, write-only?
10401260h 4    ;PICA(0098h)        unused (0)   ;XXX used, write-only?
1040127Ch 24h  ;PICA(009Fh..00A7h) unused (0)
104012E0h 20h  ;PICA(00B8h..00BFh) unused (0)
10401314h 08h  ;PICA(00C5h..00C6h) unused (0)           ;-TexEnv0
10401334h 0Ch  ;PICA(00CDh..00CFh) unused (0)           ;-TexEnv1
10401354h 0Ch  ;PICA(00D5h..00D7h) unused (0)           ;-TexEnv2
10401374h 0Ch  ;PICA(00DDh..00DFh) unused (0)           ;-TexEnv3
1040139Ch 4    ;PICA(00E7h)        unused (0)
104013D4h 0Ch  ;PICA(00F5h..00F7h) unused (0)           ;-TexEnv4
10401420h 14h  ;PICA(0108h..010Ch) unused (0)
104014C4h 38h  ;PICA(0131h..013Eh) unused (0)
10401520h 4    ;PICA(0148h)        unused (0)           ;\LIGHT0
10401530h 10h  ;PICA(014Ch..014Fh) unused (0)           ;/
10401560h 4    ;PICA(0158h)        unused (0)           ;\LIGHT1
10401570h 10h  ;PICA(015Ch..015Fh) unused (0)           ;/
104015A0h 4    ;PICA(0168h)        unused (0)           ;\LIGHT2
104015B0h 10h  ;PICA(016Ch..016Fh) unused (0)           ;/
104015E0h 4    ;PICA(0178h)        unused (0)           ;\LIGHT3
104015F0h 10h  ;PICA(017Ch..017Fh) unused (0)           ;/
10401620h 4    ;PICA(0188h)        unused (0)           ;\LIGHT4
10401630h 10h  ;PICA(018Ch..018Fh) unused (0)           ;/
10401660h 4    ;PICA(0198h)        unused (0)           ;\LIGHT5
10401670h 10h  ;PICA(019Ch..019Fh) unused (0)           ;/
104016A0h 4    ;PICA(01A8h)        unused (0)           ;\LIGHT6
104016B0h 10h  ;PICA(01ACh..01AFh) unused (0)           ;/
104016E0h 4    ;PICA(01B8h)        unused (0)           ;\LIGHT7
104016F0h 10h  ;PICA(01BCh..01BFh) unused (0)           ;/
10401704h 4    ;PICA(01C1h)        unused (0)           ;\
1040171Ch 4    ;PICA(01C7h)        unused (0)           ; LIGHTING
10401768h 18h  ;PICA(01DAh..01DFh) unused (0)           ;/
104018ACh 08h  ;PICA(022Bh..022Ch) unused (0)           ;-VERTEX
```

Unused command numbers PICA(0000h..000Fh, 0020h..003Fh)
Registers at 10401000h..1040103Fh and 10401080h..104010FFh are writeable by ARM11 only, not via PICA command numbers 0000h..000Fh, 0020h..003Fh.

Unused command numbers PICA(0400h..FFFFh)
The PICA command numbers in CMDBUF are said to be 16bit wide, however, values 0400h..FFFFh seem to be unused (and there are no I/O addresses for them; accessing 10402000h and up triggers data abort).

3DS GPU Shader Instruction Set - Opcode Summary

Shader Instruction Set Summary

Opcode	Fmt	Name	Description
00h	1	ADD	Add two vectors component by component
01h	1	DP3	Dot product on two 3-component vectors
02h	1	DP4	Dot product on two 4-component vectors
03h	1	DPH	Dot product on a 3-component and a 4-component vector
04h	1	DST	Equivalent to Microsoft's "dst" instruction
05h	1u	EX2	Exponent with base 2 on 1st component of SRC1
06h	1u	LG2	Logarithm with base 2 on 1st component of SRC1

```
07h     1u   LITP    Related to Microsoft's "lit" instruction
08h     1    MUL     Multiply two vectors component by component
09h     1    SGE     Set output if SRC1>=SRC2
0Ah     1    SLT     Set output if SRC1<SRC2
0Bh     1u   FLR     Computes SRC1's floor component by component
0Ch     1    MAX     Max of two vectors, component by component
0Dh     1    MIN     Min of two vectors, component by component
0Eh     1u   RCP     Reciprocal of vector's 1st component
0Fh     1u   RSQ     Reciprocal of square root of vector's 1st component
10h-11h ?    ?       ?
12h     1u   MOVA    Move to address register (float to integer) (to "a0" ?)
13h     1u   MOV     Move to register
14h-17h ?    ?       ?
18h     1i   DPHI    Dot product on a 3-component and a 4-component vector
19h     1i   DSTI    DST with sources swapped
1Ah     1i   SGEI    Set output if SRC1>=SRC2
1Bh     1i   SLTI    Set output if SRC1<SRC2
1Ch-1Fh ?    ?       ?
20h     0    BREAK   Break out of LOOP block
21h     0    NOP     No operation
22h     0    END     Done (stop execution and set ready flag?)
23h     2    BREAKC  Break (if condition is true)
24h     2    CALL    Call (jump to DST, and return after NUM instructions)
25h     2    CALLC   Call (as above, if condition is true)
26h     3    CALLU   Call (as above, if BOOL is true)
27h     3    IFU     Jump if/else BOOL is true
28h     2    IFC     Jump if/else condition is true
29h     3    LOOP    Loop start (repeat following opcodes)
2Ah     0    EMIT    Geometry shader only: Emit vertex (or vertex+primitive)
2Bh     4    SETEMIT Geometry shader only: Configure type/flags for EMIT
2Ch     2    JMPC    Jump if condition is true
2Dh     3    JMPU    Jump if BOOL is true
2Eh-2Fh 1c   CMP     Compare X and Y components and Set booleans cmp.x/y
30h-37h 5i   MADI    Multiply and Add vectors, component by component
38h-3Fh 5    MAD     Multiply and Add vectors, component by component
```

Shader Opcode/Parameter Encoding Formats

```
|..3 ..................2 ..................1 ..................0|
|1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0| Fmt
|_Opcode____|__?_____________________________________________________| 0  ;NoParam
|_Opcode____|__Dst____|i1_|__Src1_______|__Src2___|______Desc___| 1  ;\
|_Opcode____|__Dst____|i2_|__Src1___|______Src2___|______Desc___| 1i ; misc
|_Opcode____|__Dst____|i1_|__Src1_______|__?______|______Desc___| 1u ;/
|_Opcode__|CmpX_|CmpY_|i1_|__Src1_______|__Src2___|______Desc___| 1c CMP
|_Opcode____|X|Y|Cnd|________DstOffs________|_?_|__NumInstr_____| 2  ;\JMP
|_Opcode____|uID____|________DstOffs________|_?_|__NumInstr?____| 3  ;/etc.
|_Opcode____|vID|P|W|__?_____________________________________________| 4  ;SETEMIT
|_Opc_|__Dst____|i2_|__Src1___|__Src2_______|__Src3___|__Desc___| 5  ;MAD
|_Opc_|__Dst____|i3_|__Src1___|__Src2___|______Src3___|__Desc___| 5i ;MADI
```

Shader Operand descriptors (OPDESC's)

```
|..3 ..................2 ..................1 ..................0|
|1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0|
| |s0 |s1 |s2 |s3 |n|s0 |s1 |s2 |s3 |n|s0 |s1 |s2 |s3 |n|X_Y_Z_W|
   <---Source3-----> <---Source2-----> <---Source1-----> <-Dst->
```

3DS GPU Shader Instruction Set - Blurp

Overview

A compiled shader binary is comprised of two parts: the main instruction sequence and the operand descriptor table. These are both sent to the GPU around the same time but using separate GPU Commands. Instructions

(such as format 1 instruction) may reference operand descriptors. When such is the case, the operand descriptor ID is the offset, in words, of the descriptor within the table. Both instructions and descriptors are coded in little endian. Basic implementations of the following specification can be found at [1] and [2]. The instruction set seems to have been heavily inspired by Microsoft's vs_3_0 [3] and the Direct3D shader code [4]. Please note that this page is being written as the instruction set is reverse engineered; as such it may very well contain mistakes.

Debug information found in the code.bin of "Ironfall: Invasion" suggests that there may not be more than 512 instructions and 128 operand descriptors in a shader.

Nomenclature

```
 - opcode names with I appended to them are the same as their non-I version,
     except they use the inverted instruction format, giving 7 bits to SRC2
     (and access to uniforms) and 5 bits to SRC1
 - opcode names with U appended to them are the same as their non-U version,
     except they are executed conditionally based on the value of a uniform
     boolean.
 - opcode names with C appended to them are the same as their non-C version,
     except they are executed conditionally based on a logical expression
     specified in the instruction.
```

Instruction formats

```
  Offset  Size (bits)     Description
 Format 1: (used for register operations)
  0x0  0-6    0x7     Operand descriptor ID (DESC)
  0x7  7-11   0x5     Source 2 register (SRC2)
  0xC  12-18  0x7     Source 1 register (SRC1)
  0x13 19-20  0x2     Address register index for SRC1 (IDX_1)
  0x15 21-25  0x5     Destination register (DST)
  0x1A 26-31  0x6     Opcode
 Format 1i: (used for register operations)
  0x0  0-6    0x7     Operand descriptor ID (DESC)
  0x7  7-13   0x7     Source 2 register (SRC2)
  0xE  14-18  0x5     Source 1 register (SRC1)
  0x13 19-20  0x2     Address register index for SRC2 (IDX_2)
  0x15 21-25  0x5     Destination register (DST)
  0x1A 26-31  0x6     Opcode
 Format 1u: (used for unary register operations)
  0x0  0-6    0x7     Operand descriptor ID (DESC)
   7   7-11     5     ?
  0xC  12-18  0x7     Source 1 register (SRC1)
  0x13 19-20  0x2     Address register index for SRC1 (IDX_1)
  0x15 21-25  0x5     Destination register (DST)
  0x1A 26-31  0x6     Opcode
 Format 1c: (used for comparison operations)
  0x0  0-6    0x7     Operand descriptor ID (DESC)
  0x7  7-11   0x5     Source 2 register (SRC2)
  0xC  12-18  0x7     Source 1 register (SRC1)
  0x13 19-20  0x2     Address register index for SRC1 (IDX_1)
  0x15 21-23  0x3     Comparison operator for Y (CMPY)
  0x18 24-26  0x3     Comparison operator for X (CMPX)
  0x1B 27-31  0x5     Opcode (5bit only)
 Format 2: (used for flow control instructions)
  0x0  0-7    0x8     Number of instructions (NUM)
    8  8-9    ?       ?
  0xA  10-21  0xC     Destination offset (in words) (DST)
  0x16 22-23  0x2     Condition boolean operator (CONDOP)
  0x18 24     0x1     Y reference bit (REFY)
  0x19 25     0x1     X reference bit (REFX)
  0x1A 26-31  0x6     Opcode
 Format 3: (used for uniform-based conditional flow control instructions)
  0x0  0-7    0x8     Number of instructions ? (NUM)
```

```
   8  8-9    ?       ?
 0xA  10-21  0xC     Destination offset (in words) (DST)
 0x16 22-25  0x4     Uniform ID (BOOL/INT)
 0x1A 26-31  0x6     Opcode
Format 4: (used for SETEMIT)
   0  0-21   ?       ?
 0x16 22     0x1     Winding flag (FLAG_WINDING)
 0x17 23     0x1     Primitive emit flag (FLAG_PRIMEMIT)
 0x18 24-25  0x2     Vertex ID (VTXID)
 0x1A 26-31  0x6     Opcode
Format 5: (used for MAD)
 0x0  0-4    0x5     Operand descriptor ID (DESC)
 0x5  5-9    0x5     Source 3 register (SRC3)
 0xA  10-16  0x7     Source 2 register (SRC2)
 0x11 17-21  0x5     Source 1 register (SRC1)
 0x16 22-23  0x2     Address register index for SRC2 (IDX_2)
 0x18 24-28  0x5     Destination register (DST)
 0x1D 29-31  0x3     Opcode (3bit only)
Format 5i: (used for MADI)
 0x0  0-4    0x5     Operand descriptor ID (DESC)
 0x5  5-11   0x7     Source 3 register (SRC3)
 0xC  12-16  0x5     Source 2 register (SRC2)
 0x11 17-21  0x5     Source 1 register (SRC1)
 0x16 22-23  0x2     Address register index for SRC3 (IDX_3)
 0x18 24-28  0x5     Destination register (DST)
 0x1D 29-31  0x3     Opcode (3bit only)
```

Instructions

Unless noted otherwise, SRC1 and SRC2 refer to their respectively indexed float[4] registers (after swizzling). Similarly, DST refers to its indexed register modulo destination component masking, i.e. an expression like DST=SRC1 might actually just set DST.y to SRC1.y.

```
 Opcode  Format Name    Description
 00h     1      ADD     Adds two vectors component by component
                              DST[i] = SRC1[i]+SRC2[i] for all i
 01h     1      DP3     Computes dot product on 3-component vectors
                              DST = SRC1.SRC2
 02h     1      DP4     Computes dot product on 4-component vectors
                              DST = SRC1.SRC2
 03h     1      DPH     Computes dot product on a 3-component vector
                        with 1.0 appended to it and a 4-component vector
                              DST = SRC1.SRC2 (with SRC1 homogenous)
 04h     1      DST     Equivalent to Microsoft's dst instruction
                              DST = {1, SRC1[1]*SRC2[1], SRC1[2], SRC2[3]}
 05h     1u     EX2     Computes SRC1's first component exponent with base 2
                              DST[i] = EXP2(SRC1[0]) for all i
 06h     1u     LG2     Computes SRC1's first component logarithm with base 2
                              DST[i] = LOG2(SRC1[0]) for all i
 07h     1u     LITP    Appears to be related to Microsoft's lit instruction
                              DST = clamp(SRC1, min={0, -127.9961, 0, 0},
                                    max={inf, 127.9961, 0, inf})
                              ;note: 127.9961 = 0x7FFF / 0x100
 08h     1      MUL     Multiplies two vectors component by component
                              DST[i] = SRC1[i].SRC2[i] for all i
 09h     1      SGE     Sets output if SRC1 is greater than or equal to SRC2
                              DST[i] = (SRC1[i] >= SRC2[i]) ? 1.0 : 0.0
                              "for all i"    ;uh, who/what is "all i"?
 0Ah     1      SLT     Sets output if SRC1 is strictly less than SRC2
                              DST[i] = (SRC1[i] < SRC2[i]) ? 1.0 : 0.0
                              "for all i"
 0Bh     1u     FLR     Computes SRC1's floor component by component
                              DST[i] = FLOOR(SRC1[i])
                              "for all i"
 0Ch     1      MAX     Takes the max of two vectors, component by component
                              DST[i] = MAX(SRC1[i], SRC2[i])
```

			"for all i"
0Dh	1	MIN	Takes the min of two vectors, component by component DST[i] = MIN(SRC1[i], SRC2[i]) "for all i"
0Eh	1u	RCP	Computes the reciprocal of the vector's 1st component DST[i] = 1/SRC1[0] "for all i"
0Fh	1u	RSQ	Computes the reciprocal of the square root of the vector's first component; DST[i] = 1/sqrt(SRC1[0]) "for all i"
10h-11h	?	???	?
12h	1u	MOVA	Move to address register; Casts the float uniform given by SRC1 to an integer (truncating the fractional part) and assigns the result to (a0.x, a0.y, _, _), respecting the destination component mask.
13h	1u	MOV	Moves value from one register to another DST = SRC1
14h-17h	?	???	?
18h	1i	DPHI	Computes dot product on a 3-component vector with 1.0 appended to it and a 4-component vector; DST = SRC1.SRC2 (with SRC1 homogenous)
19h	1i	DSTI	DST with sources swapped
1Ah	1i	SGEI	Sets output if SRC1 is greater than or equal to SRC2 DST[i] = (SRC1[i] >= SRC2[i]) ? 1.0 : 0.0 "for all i"
1Bh	1i	SLTI	Sets output if SRC1 is strictly less than SRC2 DST[i] = (SRC1[i] < SRC2[i]) ? 1.0 : 0.0 "for all i"
1Ch-1Fh	?	???	?
20h	0	BREAK	Breaks out of LOOP block; do not use while in nested IF/CALL block inside LOOP block.
21h	0	NOP	Does literally nothing.
22h	0	END	Signals the shader unit that processing for this vertex/primitive is done.
23h	2	BREAKC	If condition (see below for details) is true, then breaks out of LOOP block.
24h	2	CALL	Jumps to DST and executes instructions until it reaches DST+NUM instructions
25h	2	CALLC	If condition (see below for details) is true, then jumps to DST and executes instructions until it reaches DST+NUM instructions, else does nothing.
26h	3	CALLU	Jumps to DST and executes instructions until it reaches DST+NUM instructions if BOOL is true
27h	3	IFU	If condition BOOL is true, then executes instructions until DST, then jumps to DST+NUM; else, jumps to DST.
28h	2	IFC	If condition (see below for details) is true, then executes instructions until DST, then jumps to DST+NUM; else, jumps to DST
29h	3	LOOP	Loops over the code between itself and DST (inclusive), performing INT.x+1 iterations in total. First, aL is initialized to INT.y. After each iteration, aL is incremented by INT.z.
2Ah	0	EMIT	(geometry shader only) Emits a vertex (and primitive if FLAG_PRIMEMIT was set in the corresponding SETEMIT). SETEMIT must be called before this.
2Bh	4	SETEMIT	(geometry shader only) Sets VTXID, FLAG_WINDING and FLAG_PRIMEMIT for the next EMIT instruction. VTXID is the ID of the vertex about to be emitted within the primitive, while FLAG_PRIMEMIT is zero if we are just emitting a single vertex and non-zero if are emitting a vertex and primitive simultaneously. FLAG_WINDING controls the output primitive's winding. Note that the output vertex buffer (which holds 4 vertices) is not cleared when the primitive is

```
                        emitted, meaning that vertices from the previous
                        primitive can be reused for the current one. (this
                        is still a working hypothesis and unconfirmed)
  2Ch     2      JMPC   If condition (see below for details) is true, then
                        jumps to DST, else does nothing.
  2Dh     3      JMPU   If condition BOOL is true, then jumps to DST, else
                        does nothing. Having bit 0 of NUM = 1 will invert the
                        test, jumping if BOOL is false instead.
  2Eh-2Fh 1c     CMP    Sets booleans cmp.x and cmp.y based on the operand's
                        x and y components and the CMPX and CMPY comparison
                        operators respectively. See below for details about
                        operators. It's unknown whether CMP respects the
                        destination component mask or not.
  30h-37h 5i     MADI   Multiplies two vectors and adds a third one
                        component by component;
                              DST[i] = SRC3[i] + SRC2[i].SRC1[i]
                              "for all i"
                              "this is not an FMA, the intermediate result is
                              rounded"
  38h-3Fh 5      MAD    Multiplies two vectors and adds a third one
                        component by component;
                              DST[i] = SRC3[i] + SRC2[i].SRC1[i]
                              "for all i"
                              "this is not an FMA, the intermediate result
                              is rounded"
```

Shader Operand descriptors (OPDESC's)

Below 31bit (or 32bit?) are related to the 5bit/7bit "DESC" id's in opcode LSBs.

```
  0      Destination component mask W (0=Don't change, 1=Write)
  1      Destination component mask Z (0=Don't change, 1=Write)
  2      Destination component mask Y (0=Don't change, 1=Write)
  3      Destination component mask X (0=Don't change, 1=Write)
  4      Source 1 Negation bit
  5-6    Source 1 Component 3 value   (0..3 = X,Y,Z,W) (usually 3)
  7-8    Source 1 Component 2 value   (0..3 = X,Y,Z,W) (usually 2)
  9-10   Source 1 Component 1 value   (0..3 = X,Y,Z,W) (usually 1)
  11-12  Source 1 Component 0 value   (0..3 = X,Y,Z,W) (usually 0)
  13     Source 2 Negation bit
  14-15  Source 2 Component 3 value   (0..3 = X,Y,Z,W)
  16-17  Source 2 Component 2 value   (0..3 = X,Y,Z,W)
  18-19  Source 2 Component 1 value   (0..3 = X,Y,Z,W)
  20-21  Source 2 Component 0 value   (0..3 = X,Y,Z,W)
  22     Source 3 Negation bit
  23-24  Source 3 Component 3 value   (0..3 = X,Y,Z,W)
  25-26  Source 3 Component 2 value   (0..3 = X,Y,Z,W)
  27-28  Source 3 Component 1 value   (0..3 = X,Y,Z,W)
  29-30  Source 3 Component 0 value   (0..3 = X,Y,Z,W)
  31     unnown/unspecified/unused (usually 0)
```

The component selector (the 8bit fragments with 4x2bits) enables swizzling. For example, component selector 0x1B is equivalent to .xyzw, while 0x55 is equivalent to .yyyy.

Depending on the current shader opcode, source components are disabled implicitly by setting (*to ZERO*) the destination component mask. For example, ADD o0.xy, r0.xyzw, r1.xyzw will not make use of r0's or r1's z/w components, while DP4 o0.xy, r0.xyzw, r1.xyzw will use all input components regardless of the used destination component mask.

Relative addressing

```
  IDX  Address offset
  00h  +0       ;-no offset
  01h  +a0.x    ;\X or Y component of address register (see MOVA)
  02h  +a0.y    ;/
  03h  +lp      ;-loop counter (see LOOP)
```

There are 3 address registers: a0.x, a0.y and aL (loop counter). For format 1 instructions, when IDX != 0, the value of the corresponding address register is added to SRC1's value. For example, if IDX = 2, a0.y = 3 and SRC1 = c8, then instead SRC1+a0.y = c11 will be used for the instruction. It is only possible to use address registers with vector uniform registers, attempting to use them with input attribute or temporary registers results in the address register being ignored (i.e. read as zero).

a0.x and a0.y are set manually through the MOVA instruction by rounding a float value to integer precision. Hence, they may take negative values.

aL can only be set indirectly by the LOOP instruction. It is still accessible and valid after exiting a LOOP block, though.

Comparison operator

CMPX/CMPY raw value	Operator name	Expression
00h	EQ	src1 == src2
01h	NE	src1 != src2
02h	LT	src1 < src2
03h	LE	src1 <= src2
04h	GT	src1 > src2
05h	GE	src1 >= src2
06h	??	seems to always return true
07h	??	seems to always return true

Conditions

A number of format 2 instructions are executed conditionally. These conditions are based on two boolean registers which can be set with CMP: cmp.x and cmp.y.

Conditional instructions include 3 parameters: CONDOP, REFX and REFY. REFX and REFY are reference values which are tested for equality against cmp.x and cmp.y, respectively. CONDOP describes how the final truth value is constructed from the results of the two tests. There are four conditional expression formats:

CONDOP raw value	Expression	Description
00h	cmp.x == REFX || cmp.y == REFY	OR
01h	cmp.x == REFX && cmp.y == REFY	AND
02h	cmp.x == REFX	X
03h	cmp.y == REFY	Y

Registers

Input attribute registers (v0-v7?) store the per-vertex data given by the CPU and hence are read-only.
Output attribute registers (o0-o6) hold the data to be passed to the later GPU stages and are write-only. Each of the output attribute register components is assigned a semantic by setting the corresponding GPU_Internal_Registers.
Uniform registers hold user-specified data which is constant throughout all processed vertices. There are 96 float[4] uniform registers (c0-c95), eight(???) boolean registers (b0-b7), and four int[4] registers (i0-i3).
Temporary registers (r0-r15) can be used for intermediate calculations and can both be read and written.
Many shader instructions which take float arguments have only 5 bits available for the second argument. They may hence only refer to input attributes or temporary registers. In particular, it's not possible to pass two float[4] uniforms to these instructions.
It appears that writing twice to the same output register can cause problems (eg. GPU hangs).

DST mapping:

DST raw value	Register name	Description
00h-06h?	o0-o6?	Output registers (aka output FIFO) (W)
10h-1Fh	r0-r15	Temporary registers (R/W)

SRC1/SRC2/SRC3 mapping:

SRC raw value	Register name	Description
00h-07h?	v0-v7?	Input vertex/attribute (aka input FIFO) (R)

```
10h-1Fh        r0-r15          Temporary registers
20h-7Fh        c0-c95          Vector uniform registers (only for 7bit SRC)
```

Control Flow
Control flow is implemented using four independent stacks:
```
4-deep CALL stack
8-deep IF stack
4-deep LOOP stack
```
All stacks are initially empty. After every instruction but before JMP takes effect, the PC is incremented and a copy is sent to each stack. Each stack is checked against its copy of the PC. If an entry is popped from the stack, the copied PC is updated and used for the next check of this stack, although the IF/LOOP stacks can each only pop one entry per instruction, whereas the CALL stack is checked again until it doesn't match or the stack is empty. The updated PC copy with the highest priority wins: LOOP (highest), IF, CALL, JMP, original PC (lowest).

3DS GPU Geometry Pipeline

Array Attributes (via ARRAY pointers)
The most common way seems to be drawing via GPUREG_ATTR_BUF_xxx and GPUREG_ATTR_BUF_DRAWARRAYS, the GPU is then automatically reading vertex data from the array pointers (instead of needing to send all vertices to the command buffer).

Fixed Vertex Attributes (fixed attr for ALL vertices)
If a certain vertex attribute is constant for the duration of a draw call, instead of specifying a vertex array with repeated contents or changing the shader to use a uniform, fixed vertex attributes can be used. They let you specify a fixed value, which will be assumed by the attribute for all vertices of the batch.

To use a fixed attribute, set the bit corresponding to the attribute in GPUREG_ATTR_BUF_FORMAT_HIGH and ensure that no vertex arrays are configured for the attribute (any configured arrays will override the fixed value, regardless of the bit setting). Even if a vertex array isn't being used for the attribute it still needs to be counted in the number of active attributes specified in the same register.

To specify the actual value of the fixed attribute, write the attribute index to GPUREG_FIXEDATTRIB_INDEX followed by writes with packed a float24 4-tuple to the 3 GPUREG_FIXEDATTRIB_DATA registers. The value is always specified as a float 4-component vector, the configured type is ignored.

Immediate-Mode Vertex Submission (via GPUREG_FIXEDATTRIB_DATA)
Instead of using vertex arrays to supply vertex data, drawing can be done by directly writing vertex data to a register. This allows vertex data to be inlined directly in the command buffer. Since this is restricted to 4-component float data, it is more useful for small draws like UI elements or debug displays, to avoid using an unreasonable amount of memory and processing time appending the vertices to the command buffer.

To use this feature, configure the number of attributes per vertex in GPUREG_VSH_NUM_ATTR. (All settings in the registers related to the vertex loader are ignored.) Then setup the GPU and shaders the same as if doing a regular draw call with GPUREG_ATTR_BUF_DRAWARRAYS or GPUREG_ATTR_BUF_DRAWELEMENTS, but instead of writing to either register, write the value 0xF to GPUREG_FIXEDATTRIB_INDEX and then follow by repeatedly writing vertex data to GPUREG_FIXEDATTRIB_DATA.

Each set of writes to the 3 data registers specifies one attribute and all attributes (as configured in GPUREG_VSH_NUM_ATTR) need to be written, in order, to specify a vertex. Drawing happens automatically as vertices are specified. After finishing specifying vertices, follow with the same writes used after a draw arrays/elements.

When drawing using triangle strips or fans, GPUREG_RESTART_PRIMITIVE should be used to end the previous strip before (or while) drawing.

Drawing elements
The 3DS GPU is capable of drawing vertex + index arrays, triggered by GPUREG_ATTR_BUF_DRAWELEMENTS. A set of commands commonly used by the standard GL implementation to accomplish this is as follows:

CommandIndex	Register	Description
0	GPUREG_GEOSTAGE_CONFIG	Set whether drawing triangle elements
1-2	GPUREG_GEOSTAGE_CONFIG2	Set whether drawing triangle elements
3	GPUREG_PRIMITIVE_CONFIG	Set primitive mode
4	GPUREG_PRIMITIVE_CONFIG	Set number of output map registers
5	GPUREG_RESTART_PRIMITIVE	Trigger reset
6	GPUREG_GEOSTAGE_CONFIG2	Set function indicator to 0
7	GPUREG_ATTR_BUF_INDEX_LIST	Set offset and type
8	GPUREG_ATTR_BUF_NUMVERTICES	Set vertex count (mul2 supposedly?)
9	GPUREG_START_DRAW_FUNC0	Set mode to drawing
10	GPUREG_ATTR_BUF_DRAWELEMENTS	Trigger draw
11	GPUREG_START_DRAW_FUNC0	Set mode to configuration
12	GPUREG_VTX_FUNC	Trigger post-vertex cache clear
13	GPUREG_RENDERBUFFER_FLUSH	Flush framebuffer
14	GPUREG_GEOSTAGE_CONFIG	Clear drawing triangle elements
15	GPUREG_GEOSTAGE_CONFIG2	Clear drawing triangle elements
16	GPUREG_PRIMITIVE_CONFIG	Clear primitive mode, uh?
17	GPUREG_VSH_ENTRYPOINT	Clear entry point, huh?

Uh, the GPU is NOT automatically doing that! Above is just a rather incomplete flowchart for manually setting the above registers to unspecifed values.

3DS GPU Fragment Lighting

Fragment lighting is a DMP extension to the standard OpenGL pipeline with which applications can calculate object lighting for each rendered pixel instead of just per vertex. The fragment lighting algorithm furthermore supports the shading models Blinn-Phong, Cook-Terrance, Ward, and microfacet-based BRDF-models. While the lighting calculations take place in a very localized position of the pixel processing pipeline, the feature interacts with different other pipeline stages.

Overview
In general, lighting is calculated at a particular point in space X by determining the angles (i.e. dot products) between different vectors:

- The face normal vector N, which points from X to the direction perpendicular to the lighted object
- The tangent vector T, which points from X to a direction which is tangential to the lighted object
- The bitangent vector B, which points to a tangential direction such that it is orthogonal to both N and T
- The view vector V, which points from X "into the camera"
- The light vector L, which points from X to the light source (note that lighting is always evaluated separately for each light source; the results for multiple light sources can simply be added to each other)

For example, in the Blinn-Phong shading model the dot product of L and N determined the intensity of a lighting type called diffuse lighting. More generally, the dot products between these vectors (from now on simply referred to as "dot products") are combined to a lighting equation, which is evaluated once per light source at each considered point in space.

Before pixel shaders were common, the lighting equation was only considered at each vertex, and the output color was then interpolated across the triangle. To achieve higher visual quality, pixel shaders can be used

nowadays to evaluate the lighting equation at each pixel. The PICA200 does not have a programmable pixel shader, but has a fixed-function pipeline stage to achieve the same quality.

In any case, per-pixel lighting requires to somehow obtain a normal vector for each pixel. Theoretically, the vertex shader could output one normal vector per vertex, and per-pixel normals could then be obtained by interpolating these vectors in a specific way. This, however, is very inefficient, which is why vertex shaders on the PICA200 instead output mathematical objects called quaternions: Quaternions can be computed from normal/tangent vectors, and vice versa, so no information is lost by doing so. However, quaternion interpolation is a lot faster (see the "Kuijk and Blake" source below), and so the GPU can compute per-pixel normals simply using the interpolated quaternion. This is just the general idea, and the dirty mathematical details are explained below.

In addition to allowing per-pixel calculations, the lighting equation used in the PICA200 involves using the dot products as indices into configurable lookup tables. A good example of why this is useful is cell-shading, which can be achieved simply by setting adjacent lookup table values to the same color.

Quaternions as Shader Outputs
The shaders are usually the point where normal and tangent vector information flows in. Usually, the source vertex data will include normal vectors for each vertex. This need not be the case though; one could also just input raw vertex positions, and have a geometry shader automatically calculate normal vector information.

In any case: There is no vertex output attribute semantic for normal or tangent vectors. To use fragment lighting, the shader must actually output an attribute with the quaternion semantic. So some sort of conversion needs to happen from normal and tangent vectors to quaternions. This can be done using the surface-local matrix.

Quaternion Interpolation
Quaternion interpolation needs to happen to obtain a per-fragment quaternion, from which in turn per-fragment normals and tangent vectors can be computed. It is unknown how quaternions are interpolated on the PICA200. The architecture suggests that plain linear interpolation is used, but there are also more sophisticated algorithms like Slerp and Nlerp.

Quaternions and Normal/Tangent Vectors
Quaternions describe a transformation from surface-local space to eye space. In surface-local space, by definition (and up to permutation) the normal vector is (0,0,1), the tangent vector is (1,0,0), and the bitangent vector is (0,1,0).

Alternatively, one may consider quaternions a transformation from eye space to surface-local space.

Fragment Lighting Equation
There are two lighting equations: One for the primary color and one for the secondary color. Both of them are given in section 3.2.2 of the "Kazakov and Ohbuchi" source below. In addition, the fragment lighting can be set up to write to the alpha channel of the primary or secondary color depending on a selectable angle and a LUT. The equations used can be found here: http://mathb.in/26766

Some common setups include:

```
  ------------------------------------------------
  Model:            Blinn-Phong
  LutD0:            Input: N dot H, Contents: x^s
  LutD1:            --
  ReflectionLuts:   --
  Geometry factors: Disabled
  SpotlightLut      Spotlight setup or no-op
  ------------------------------------------------
  Model:            Cook-Torrance
  LutD0:            --
  LutD1:            Input: N dot H, D(x)
```

```
ReflectionLuts:   Input: V dot H, Contents: F(x)
Geometry factors: Enabled
SpotlightLut      Spotlight setup or no-op
-------------------------------------------------
Model:            Schlick-like
LutD0:            --
LutD1:            Input: N dot H, Contents: Schlick Z(x)
ReflectionLuts:   Input: V dot H, Contents: F(x)
Geometry factors: Enabled
SpotlightLut      Input: cos phi_i, Contents: Schlick A(x)
-------------------------------------------------
Model:            Subsurface scattering (?)
LutD0:            --
LutD1:            Input: N dot V, Contents: Transmittance by angle
ReflectionLuts:   Input: L dot N, Contents: Reflectance by angle
Geometry factors: Disabled
SpotlightLut      Spotlight setup or no-op
-------------------------------------------------
```

s is the specularity factor for Blinn-Phong.

Spotlight setup means input -L dot P and contents spotlight falloff.

F and D can be found in http://inst.cs.berkeley.edu/~cs294-13/fa09/lectures/cookpaper.pdf

Schlick Z and Schlick A are defined in
http://www.cs.virginia.edu/~jdl/bib/appearance/analytic%20models/schlick94b.pdf

The Fresnel LUT can be used to, for example, blend two colours according to how oblique the view angle is, or to simply additively blend white onto fragments with an exponential falloff, resulting in rim lighting.

Fragment Lighting Output
The fragment lighting results are accessible as two inputs to the texture combiners (one for the primary color, one for the secondary color).

Resources
This wiki page can only give a short overview of the fragment lighting feature. Luckily, there is a plethora of public literature available which describes the feature in more detail:

```
- Everitt - "Per-Pixel Lighting": A presentation given at the Game
  Developers Conference 2001 about per-pixel lighting. It doesn't have
  anything to do with the PICA200 algorithm, but explains the core ideas
  very well (especially the concepts of surface-local space and how it
  relates to other coordinate systems).
- Kazakov and Ohbuchi - "Primitive Processing and Advanced Shading
  Architecture for Embedded Space": Provides a general overview over the
  fragment lighting algorithm used by the PICA200 and provides explicit
  formulas for the primary and secondary lighting output. NOTE: There exist
  both a scientific publication and a short presentation with this title.
  Both are useful, but the former goes into much greater detail.
- Ohbuchi and Unno - "A Real-Time Configurable Shader Based on Lookup
  Tables": Provides a very detailed explanantion of the fragment lighting
  implementation
- Kuijk and Blake - "Faster Phong Shading via Angular Interpolation":
  Explains in greater detail how quaternions can be used to encode
  information about normals and tangents (and how quaternions are easier to
  interpolate than vectors).
```

3DS GPU Procedural Texture Generation

```
NOTE: Related registers are GPUREG_TEXUNIT3_PROCTEX0..5
```

Overview
Procedural texture generation has four stages:
```
Noise Module (outputs u',v')
Repeat Module (outputs u'',v'')
Base Shape (also notated as G(u'',v''), output g)
F(g) and Lookup Table
```

Noise Module
This stage applies noise on the input coordinates. Little is known about this other than that there are three noise parameters:
```
Amplitude
Frequency
Phase
```

Repeat Module
This stage performs basic texture coordinate wrapping on the noised coordinates. It supports symmetric and mirrored wrapping. They don't seem to be configurable beyond that.

Base Shape
The U" and V" coordinates are used to generate a scalar value in the range [0;1] from the wrapped coordinates using one of six functions.
The output of this function is named "g".

F(g) and Lookup Table
F is a selectable function which transforms g to another scalar value. There are two known options for F:
```
the identity function
a triangle function
```
The final texel color is determined by using the value of F(g) as an index into a configurable lookup table.

3DS GPU Pitfalls

Vertex attribute alignment
Vertex components which are defined through GPUREG_ATTR_BUFi_CONFIG_LOW will be accessed aligned by the GPU.
Vertex attributes will be aligned to their component element size.
Padding attributes (Component type > 11) will always aligned to 4 byte offets into the buffer.
The stride which is passed to the GPU should be passed unaligned.

Vertex stride in GPUREG_ATTR_BUFi_CONFIG_HIGH
The vertex stride set in GPUREG_ATTR_BUFi_CONFIG_HIGH must match the actual size of the vertex contained in the buffer or the PICA will freeze or it won't draw anything.
If you want to use a different stride you have to pad the data accordingly with padding attributes.

Output mapping in GPUREG_SH_OUTMAP_MASK
The output masking in GPUREG_SH_OUTMAP_MASK influences how the registers starting at GPUREG_SH_OUTMAP_Oi map to outputs in the shader.

If an output is disabled in GPUREG_SH_OUTMAP_MASK it means that no slot in the GPUREG_SH_OUTMAP_Oi registers is consumed. GPUREG_SH_OUTMAP_TOTAL configures the number of used consecutive slots in the outmap. Example:
```
GPUREG_SH_OUTMAP_TOTAL = 0x00000002      ;2 outputs enabled
GPUREG_SH_OUTMAP_MASK  = 0x00000011      ;o0 enabled, o4 enabled
GPUREG_SH_OUTMAP_O0    = 0x03020100      ;o0 = pos.xyzw
```

```
GPUREG_SH_OUTMAP_O1    = 0x0B0A0908      ;o4 = color.rgba    ;<-- o4 in "O1"!
GPUREG_SH_OUTMAP_O2    = ...             ;(unused)
```

Shaders - Configued Output components must be written exactly once
Each configured output component has to be written exactly once or the PICA freezes.

Shaders - MOVA instructions can't be adjacent
Having 2 consecutive MOVA instructions will freeze the PICA. This can be relaxed by placing a NOP between 2 MOVAs or by rearranging the code.

Shader - Special cases
JMP overwrites the PC *after* the stacks checks (and only if no stack was popped).
Executing a BREAK on an empty LOOP stack hangs the GPU.
A stack overflow discards the oldest element, so you could think of it as a queue or a ring buffer.
If the CALL stack is popped four times in a row, the fourth update to its copy of the PC is missed (the third PC update will be propagated). Probably a hardware bug.

Shader - Floating-Point Behavior
The PICA200 is not IEEE-compliant. It has positive and negative infinities and NaN, but does not seem to have negative 0. Input and output subnormals are flushed to +0. The internal floating point format seems to be the same as used in shader binaries: 1 sign bit, 7 exponent bits, 16 (explicit) mantissa bits. Several instructions also have behavior that differs from the IEEE functions. Here are the results from some tests done on hardware (s = largest subnormal, n = smallest positive normal):

```
Computation     Result  Notes
inf * 0         0       Including inside MUL, MAD, DP4, etc.
NaN * 0         NaN
+inf - +inf     NaN     Indicates +inf is real inf, not FLT_MAX
rsq(rcp(-inf))  +inf    Indicates that there isn't -0.0.
rcp(-0)         +inf    no -0 so differs from IEEE where rcp(-0) = -inf
rcp(0)          +inf
rcp(+inf)       0
rcp(NaN)        NaN
rsq(-0)         +inf    no -0 so differs from IEEE where rsq(-0) = -inf
rsq(-2)         NaN
rsq(+inf)       0
rsq(-inf)       NaN
rsq(NaN)        NaN
max(0, +inf)    +inf
max(0, -inf)    -inf
max(0, NaN)     NaN     max violates IEEE but match GLSL spec
max(NaN, 0)     0
max(-inf, +inf) +inf
min(0, +inf)    0
min(0, -inf)    -inf
min(0, NaN)     NaN     min violates IEEE but match GLSL spec
min(NaN, 0)     0
min(-inf, +inf) -inf
cmp(s, 0)       false   cmp does not flush input subnormals
max(s, 0)       s       max does not flush input or output subnormals
mul(s, 2)       0       input subnormals are flushed in arithmetic
                          instructions
mul(n, 0.5)     0       output subnormals are flushed in arithmetic
                          instructions
```

1.0 can be multiplied 63 times by 0.5 until the result compares equal zero. This is consistent with a 7-bit exponent and output subnormal flushing.

Command list related
Invalid GPU command parameters including NaN floats can cause the GPU to hang, which then causes the "GSP module" to hang as well.

3DS GPU Primitive Engine and Shaders

The 3DS seems to have four "shaders", each with there own I/O ports. However, the vertex shaders seem to be used in parallel (to process 3 or 4 vertices at once... so it's unknown if one needs to initialize all sets of I/O ports separately; 3dbrew seems to have only two shaders documented (GSH and VSH), without even being aware of the VSH2 and VSH3 registers.

Vertex Shader (VSH)
The Vertex Shader is used to forward incoming vertex data (coordinates, colors, texcoords) to the drawing engine. Most commonly, the shader would be used to multiply the incoming coordinates with a Position Matrix and/or Projection Matrix.

Geometry Shader (GSH)
The Geometry Shader can be used to do something... but it appears to be unknown what, and how.
It can probably somehow receive incoming values from somewhere. And probably output values to somewhere (maybe somehow forwarding them to other GPU registers, and maybe somehow sending them as response to the CPU).
The "EMIT" opcode might output data from Geometry Shader to Vertex shader?
The "Primitive Engine" section below might refer to the geometry shader?

Primitive Engine
Primitive Engine (PE) is one of the PICA200's four vertex processor units and provides some unique features which are used to implement a geometry shader stage and variable-size primitive rendering.
The full functionality of PE is not yet understood and remains to be reverse-engineered.
Variable-size primitives are implemented by prefixing each per-primitive sequence of indices in an index array with a primitive size. This is used for various effects, for example Catmull-Clark subdivision and Loop subdivision. It is unknown how this feature is enabled specifically.

3DS GPU Triangle Drawing Sample Code

```
_____________________________ Triangle Drawing Examples _____________________________
```

gpu_draw_triangle_direct: ;simple, but requires a messy 3x24bit data format

```
  gpu_clear_colorbuf, gpu_init_framebuf, gpu_init_vertex_shader
 ;[GPUREG_GEOSTAGE_CONFIG]=((1 shl 8)+00h) ;not needed
  [GPUREG_GEOSTAGE_CONFIG2]=(0 shl 8)+01h  ;needed
 ;[GPUREG_START_DRAW_FUNC0]=00h  ;not needed, 0=draw, 1=config?
  [GPUREG_FIXEDATTRIB_INDEX]=0Fh  ;mode for directly writing vertex data...
  for i=0 to (4*2)-1
    [GPUREG_FIXEDATTRIB_DATA+0]=[@@array+i*0Ch+8] ;\write 4x24bit as 3x32bit
    [GPUREG_FIXEDATTRIB_DATA+0]=[@@array+i*0Ch+4] ; upper 32bit written first
    [GPUREG_FIXEDATTRIB_DATA+0]=[@@array+i*0Ch+0] ;/
  next i
  gpu_copy_colorbuf_to_lcd_framebuf
  ret
 @@array: ;x_____y_____z_____w_____r_____g_____b_____a_____
  .float24 -1.0  ,-1.0  ,0.0   ,1.0   ,1.0   ,0.5   ,0.5   ,1.0 ;\1st triangle
  .float24 1.0   ,0.4   ,0.0   ,1.0   ,0.5   ,1.0   ,0.5   ,0.5 ;  ;\2nd
  .float24 0.2   ,1.0   ,0.0   ,1.0   ,0.5   ,0.5   ,1.0   ,0.5 ;/ ; triangle
  .float24 1.0   ,-0.8  ,-1.0  ,1.0   ,0.5   ,0.5   ,1.0   ,0.5    ;/(of strip)
  ;note: coord.W must be 1.0 (to avoid perspective division)
```

gpu_draw_triangle_via_buffer: ;example with 1-2 arrays, optional index_list

```
@@numattr equ 2 ;configure for two attributes (coordinate and color)
 gpu_clear_colorbuf, gpu_init_framebuf, gpu_init_vertex_shader
 [GPUREG_ATTR_BUF_BASE]=((((@@base)/10h)*2)
 [GPUREG_ATTR_BUF_FORMAT_LOW]=9Ch  ;Signed8bit(x,y,z,w)+Unsigned8bit(r,g,b)
 [GPUREG_ATTR_BUF_FORMAT_HIGH]=((@@numattr-1) shl 28)+(000h shl 16)
 if @@num_arrays=1      ;both coordinate and color in one array...
  [GPUREG_ATTR_BUFi_OFFSET+0*0Ch]=(@@array-@@base)/1
  [GPUREG_ATTR_BUFi_CONFIG_LOW+0*0Ch]=76543210h
  [GPUREG_ATTR_BUFi_CONFIG_HIGH+0*0Ch]=(@@numattr shl 28)+((4+3) shl 16)+BA98h
 elseif @@num_arrays=2  ;coordinate and color in separate arrays...
  [GPUREG_ATTR_BUFi_OFFSET+0*0Ch]=(@@array0-@@base)/1
  [GPUREG_ATTR_BUFi_OFFSET+1*0Ch]=(@@array1-@@base)/1
  [GPUREG_ATTR_BUFi_CONFIG_LOW+0*0Ch]=00000000h
  [GPUREG_ATTR_BUFi_CONFIG_LOW+1*0Ch]=00000001h
  [GPUREG_ATTR_BUFi_CONFIG_HIGH+0*0Ch]=(01h shl 28)+(04h shl 16)+0000h
  [GPUREG_ATTR_BUFi_CONFIG_HIGH+1*0Ch]=(01h shl 28)+(03h shl 16)+0000h
 endif
;[GPUREG_GEOSTAGE_CONFIG] =((1 shl 8)+00h)  ;not needed
 [GPUREG_GEOSTAGE_CONFIG2]=(1 shl 8)+01h    ;needed
;[GPUREG_START_DRAW_FUNC0]=00h              ;not needed, 0=draw, 1=config?
 [GPUREG_ATTR_BUF_NUMVERTICES]=4*(1+@@use_index_list)  ;needed, array size
 if @@use_index_list    ;with index list...
   [GPUREG_ATTR_BUF_INDEX_LIST]=(0 shl 31)+(@@index_list-@@base)/1
   [GPUREG_ATTR_BUF_DRAWELEMENTS]=0   ;start drawing via above index list
 else                   ;without index_list...
   [GPUREG_ATTR_BUF_FIRST_INDEX]=0    ;first index
   [GPUREG_ATTR_BUF_DRAWARRAYS]=0     ;start drawing at above index number
 endif
 gpu_copy_colorbuf_to_lcd_framebuf
 ret
align 10h
@@base:
if @@use_index_list
  @@index_list: db 0,1,2,3   ;index to 1st,2nd,3rd,4th entries in arrays
endif
if @@num_arrays=1
 @@array: ;x_____y_____z_____w_____r_____g_____b_____a_____
  db       -7fh  ,-7fh  ,0     ,+7fh  ,0ffh  ,0     ,0          ;\1st triangle
  db       +7fh  ,+40h  ,0     ,+7fh  ,0     ,0ffh  ,0          ;  ;\2nd
  db       +20h  ,+7fh  ,0     ,+7fh  ,0     ,0     ,0ffh       ;/ ; triangle
  db       +7fh  ,-60h  ,0     ,+7fh  ,0     ,0ffh  ,0ffh          ;/of strip
elseif @@num_arrays=2
 @@array0: ;x_____y_____z_____w____
  db        -7fh  ,-7fh  ,0 -9  ,+7fh  ;\1st triangle
  db        +7fh  ,+40h  ,0 -9  ,+7fh  ;  ;\2nd
  db        +20h  ,+7fh  ,0 -9  ,+7fh  ;/ ; triangle
  db        +7fh  ,-60h  ,-7fh  ,+7fh     ;/of strip
 @@array1: ;r_____g_____b_____a_____
  db        0ffh  ,0     ,0             ;\1st triangle
  db        0     ,0ffh  ,0             ;  ;\2nd
  db        0     ,0     ,0ffh          ;/ ; triangle
  db        0     ,0ffh  ,0ffh             ;/of strip
endif

_____________________ Init and Memory Transfer Functions _____________________
```

gpu_clear_colorbuf:

```
  [GPUREG_RENDERBUFFER_INVALIDATE]=1   ;forget cache
  [GPU_MEMFILL_DST_ADDR0]=(MEMORG_COLORBUF+0)/10h*2
  [GPU_MEMFILL_DST_END0]=(MEMORG_COLORBUF+320*240*4)/10h*2
  [GPU_MEMFILL_DATA0]=11441100h
  [GPU_MEMFILL_CNT0]=00000201h
  wait until [GPU_MEMFILL_CNT0].bit0=0
  ret
```

gpu_copy_colorbuf_to_lcd_framebuf:

```
  DummyRead=[GPUREG_STAT_NUM_TRIANGLES_DISPLAYED]  ;waits until rendering done
  [GPUREG_RENDERBUFFER_FLUSH]=1   ;cache writeback
  [GPU_MEMCOPY_SRC_ADDR]=MEMORG_COLORBUF/10h*2
  [GPU_MEMCOPY_DST_ADDR]=MEMORG_SCREEN1/10h*2
  [GPU_MEMCOPY_DISPLAY_SIZE]=(240 shl 0)+(320 shl 16)
  [GPU_MEMCOPY_REMAIN_IRQ]=3FFFh  ;want IRQ flag when done
  [GPU_MEMCOPY_FLAGS]=0
  [GPU_MEMCOPY_CNT]=1
  ;caution: polling GPU_MEMCOPY_CNT can HANG the CPU at transfer end!
  ;instead: poll GPU_STAT_IRQ_FLAGS...
  wait until: [GPU_STAT_IRQ_FLAGS].bit30=1
  ret
```

gpu_init_framebuf:

```
 ;[GPUREG_RENDERBUFFER_INVALIDATE]=1   ;forget cache
  [GPUREG_FACECULLING_CONFIG]=0 ;show front+back
  [GPUREG_RENDERBUFFER_DIM_0]=(1 shl 24)+(320-1)*1000h+240
 ;[GPUREG_RENDERBUFFER_DIM_1]=(1 shl 24)+(320-1)*1000h+240
  [GPUREG_DEPTHBUFFER_LOC]=MEMORG_DEPTHBUF/40h*8
  [GPUREG_COLORBUFFER_LOC]=MEMORG_COLORBUF/40h*8
  [GPUREG_VIEWPORT_V_SCALE]=0045E000h ;240/2 ;.float24 120.0 // db 00h
  [GPUREG_VIEWPORT_H_SCALE]=00464000h ;320/2 ;.float24 160.0 // db 00h
  [GPUREG_VIEWPORT_V_STEP]=38111100h  ;2/240 ;db 00h // .float24 0.008333333
  [GPUREG_VIEWPORT_H_STEP]=37999900h  ;2/320 ;db 00h // .float24 0.006250000
  [GPUREG_VIEWPORT_XY]=(0 shl 16)+0
  [GPUREG_SCISSORTEST_MODE]=0
 ;[GPUREG_SCISSORTEST_POS1]=((0+20) shl 16)+(0+20)
 ;[GPUREG_SCISSORTEST_POS2]=((320-1-20) shl 16)+(240-1-20)
  [GPUREG_COLOR_OPERATION]=00e40100h
  [GPUREG_BLEND_FUNC]=06020000h ;raw drawing
 ;[GPUREG_BLEND_FUNC]=76760000h ;alpha blending
 ;[GPUREG_LOGIC_OP]=00h
  [GPUREG_FRAGOP_ALPHA_TEST]=00h
  [GPUREG_STENCIL_TEST]=00h
  [GPUREG_DEPTH_COLOR_MASK]=1F00h
  [GPUREG_COLORBUFFER_READING]=0Fh
  [GPUREG_COLORBUFFER_WRITING]=0Fh
  [GPUREG_DEPTHBUFFER_READING]=03h
  [GPUREG_DEPTHBUFFER_WRITING]=03h
  [GPUREG_DEPTHBUFFER_FORMAT]=03h
  [GPUREG_COLORBUFFER_FORMAT]=02h
  [GPUREG_RENDERBUFFER_BLOCK32]=0
 ;[GPUREG_EARLYDEPTH_TEST1]=0
 ;[GPUREG_EARLYDEPTH_TEST2]=0
  [GPUREG_DEPTHMAP_ENABLE]=1
  [GPUREG_DEPTHMAP_SCALE]=00bf0000h  ;far z  (-1.0)
  [GPUREG_DEPTHMAP_OFFSET]=00000000h ;near z (0.0)
  ret
```

gpu_init_vertex_shader:

```
 @@numattr equ 2 ;configure for two attributes (coordinate and color)
  [GPUREG_SH_OUTMAP_TOTAL]=@@numattr      ;vertex+color
  [GPUREG_SH_OUTMAP_O+0*4]=03020100h      ;vertex.xyzw
  [GPUREG_SH_OUTMAP_O+1*4]=0b0a0908h      ;color.rgba
  [GPUREG_SH_OUTATTR_CLOCK]=3             ;bit0=blah?, bit1=needed for colors
  [GPUREG_VSH_COM_MODE]=00h               ;GSH/VSH?
  [GPUREG_VSH_NUM_ATTR]=@@numattr-1       ;needed
  [GPUREG_VSH_OUTMAP_TOTAL1]=@@numattr-1  ;needed
 ;[GPUREG_VSH_OUTMAP_TOTAL2]=@@numattr-1  ;not needed
  [GPUREG_START_DRAW_FUNC0]=1             ;needed
  [GPUREG_VSH_INPUTBUFFER_CONFIG]=(0a0h shl 24)+@@numattr-1
  [GPUREG_PRIMITIVE_CONFIG]=(1 shl 8)+@@numattr-1
```

```
[GPUREG_RESTART_PRIMITIVE]=0  ;required for strips (even BEFORE first strip)
[GPUREG_VSH_ATTR_PERMUTATION_LOW]=076543210h  ;\input
[GPUREG_VSH_ATTR_PERMUTATION_HIGH]=0fedcba98h ;/
[GPUREG_VSH_OUTMAP_MASK]=(1 shl @@numattr)-1        ;-output enable's
[GPUREG_VSH_CODETRANSFER_INDEX]=000h/4
[GPUREG_VSH_CODETRANSFER_DATA+0]=4C000000h  ;MOV o0,v0  ;used for coord xyzw
[GPUREG_VSH_CODETRANSFER_DATA+0]=4C201000h  ;MOV o1,v1  ;used for color rgba
[GPUREG_VSH_CODETRANSFER_DATA+0]=88000000h  ;END
[GPUREG_VSH_CODETRANSFER_END]=0
[GPUREG_VSH_OPDESCS_INDEX]=000h
[GPUREG_VSH_OPDESCS_DATA+0]=0000036Fh  ;dst=xyzw, src1=xyzw (or rgba)
[GPUREG_VSH_ENTRYPOINT]=7fff0000h+(000h/4)
ret
```

3DS Video CAM Registers (Camera Input)

10120000h - CAM0 (external Right-Eye & internal Self-Facing camera) (as DSi)
10121000h - CAM1 (external Left-Eye camera) (extra 3DS camera)

```
1012x000h 4     R/W mask: 0000ef1ch  ;CAM_CNT alike DSi cameras
1012x004h 2     R   mask: 8500h      ;CAM_STAT unlike DSi (lsbs=0..500h)
1012x006h 2     R/W mask: 003fh      ;CAM_???
1012x010h 4     R/W mask: 01ff03feh  ;CAM_SOFS alike DSi cameras
1012x014h 4     R/W mask: 01ff03feh  ;CAM_EOFS alike DSi cameras
1032x000h 80h   R                    ;CAM_DAT fifo (20h-word window)
```

The ARM registers are similar (but not identical) as for DSi cameras,
DSi Cameras
The I2C bus camera init does work same as on DSi (for Aptina cameras),
DSi I2C Bus
Cameras must be enabled in CFG11_CAMERA_CNT (Port 10141224h).

1012x000h - CAM_CNT - mask: 0000ef1ch - Control (R/W)

```
0     Unknown (status, toggles on/off during transfer?)                (R)
1     Unknown (status, gets set during transfer?)                      (R)
2     Unknown (MUST be 0, else picture gets blank?)                    (R/W)
3     Unknown (0=Normal/YUV422, 1=Some other data format?)             (R/W)
4     Unknown (MUST be 1, else data transfer hangs, maybe reset?)      (R/W)
5-7   Unused (0)                                                      (-)
8     Unknown (no effect?)                                             (R/W)
9     IRQ Enable      (0=Disable, 1=Enable) (at picture end?)          (R/W)
10    Unknown (no effect?)                                             (R/W)
11    Unknown (no effect?)                                             (R/W)
12    Unused (0)                                                       (-)
13    Unknown (MUST be 0, else data transfer hangs)                    (R/W)
14    DMA Data Request enable (0=Disable, 1=Enable)                    (R/W)
15    Transfer Enable (0=Disable/AllowConfig, 1=Enable/Transfer)       (R/W)
16-31 Unused (0)                                                       (-)
```

1012x004h - CAM_STAT - Status (R)

```
0-10  Number of 8-byte units (?) in FIFO (0..500h)                     (R)
11-14 Unused? (0)                                                      (?)
15    Overrun error, deadlock   (0=Normal, 1=Overrun)                  (R)
```

Caution: There will be no further data received after overrun. To avoid overrun, be sure to read data fast enough, and disable CAM_CNT.bit15 after reading the camera bitmap.

1012x006h - CAM_??? - Unknown (R/W)

```
0-5   Unknown (no effect?) (can be set to 0..3Fh)                      (R/W)
6-14  Unused? (0)                                                      (?)
15    Clear overrun  (0=No change, 1=Clear; when CAM_CNT.bit15=0)      (W)
```

Unknown, seems to have no direct effect on the camera picture. Maybe selects timings or amount of words for DMA requests? Or maybe some bits resemble DSi's CAM_MCNT register?

1012x010h - CAM_SOFS alike DSi cameras (0..01FF03FEh) (R/W)
1012x014h - CAM_EOFS alike DSi cameras (0..01FF03FEh) (R/W)

```
0      Unused (0)                                                    (0)
1-9    X-Offset  (0..1FFh)   in words (ie. 2-pixel units)?        (R or R/W)
10-15  Unused (0)                                                    (0)
16-24  Y-Offset  (0..1FFh)   in scanlines?                         (R or R/W)
25-31  Unused (0)                                                    (0)
```

Looks same as on DSi, but the settings seem to have no effect on 3DS, ie. there appears to be no way to enable the trimming feature?

1032x000h..1032x07Fh - CAM_DAT - Data FIFO (20h words window) (R)
The FIFO read window is 20h words in size (the actual FIFO is bigger; about 500h or A00h words?). The (default) data format is same as the YUV422 format for DSi cameras:

```
0-7   First Pixel Luminance (Y)   (unsigned, 00h..FFh, FFh=white)
8-15  Both Pixels Blue (Cb aka U) (unsigned, 00h..FFh, 80h=gray)
16-23 Second Pixel Luminance (Y)  (unsigned, 00h..FFh, FFh=white)
24-31 Both Pixels Red (Cr aka V)  (unsigned, 00h..FFh, 80h=gray)
```

Caution: It is important to read words from INCREASING addresses at 1032x000h..1032x07Fh (or at least toggle address bit2 after each read) (trying to read words from FIXED address 1032x000h will return the same word endless repeated).

Camera Notes
The "camera in-use" LED for external camera is controlled via MCU (but the LED exists on DSi/3DS only, New3DS doesn't have that LED installed).
The three 3DS/New3DS cameras have the same Chip IDs (2280h) as found in the two DSi's Aptina MT9V113 cameras.
However, New3DS is said to have improved picture quality. Unknown what that means, maybe the I2C registers are simply initialized with better gain settings at software side, or maybe the camera hardware is actually improved despite of having the same chip id.
New3DS is also said to support eye-tracking, whatever that means, it is probably done using the internal camera. And New3DS is said to have an IR-LED next to the internal camera to assist eye-tracking (yet unknown if/how the YUV/RGB camera could see IR-light at all).

3DS Video Y2R Registers (YUV-to-RGBA Converter)

Y2R Registers (aka YUV-to-RGB aka YCbCr-to-RGB)

```
10102000h ARM11/ARM9  Y2R_0 Registers              ;\original Y2R unit
10302000h ARM11       Y2R_0 FIFOs                  ;/(intended for camera)
10132000h ARM11/ARM9  Y2R_1 Registers   ;\New3DS   ;\extra New3DS Y2R unit
10332000h ARM11       Y2R_1 FIFOs       ;/         ;/(intended for MVD)
```

The Y2R unit(s) are general purpose YUV-to-RGB converters for photo/video decoding. The first Y2R unit was invented for the camera, the second Y2R unit for MVD on New3DS. The two units appear to be identical.

```
10102000h/10132000h 4   Y2R_CNT                    R/W: E8C31F07h ;\
10102004h/10132004h 2   Y2R_WIDTH   ;width (pix)   R/W: 03F8h     ;
10102006h/10132006h 2   Y2R_HEIGHT  ;height (pix)  R/W: 03FFh     ;
10102008h/10132008h ??  Y2R_STROBE  ;ack fifo?     dummy r/w?     ;
10102010h/10132010h 2   Y2R_MULTIPLIER_Y_TO_RGB    R/W: 03FFh     ; Control
10102012h/10132012h 2   Y2R_MULTIPLIER_V_TO_R      R/W: 03FFh     ; Regs
10102014h/10132014h 2   Y2R_MULTIPLIER_V_TO_G      R/W: 03FFh     ;
10102016h/10132016h 2   Y2R_MULTIPLIER_U_TO_G      R/W: 03FFh     ;
10102018h/10132018h 2   Y2R_MULTIPLIER_U_TO_B      R/W: 03FFh     ;
1010201Ah/1013201Ah 2   Y2R_OFFSET_R  ;\           R/W: FFFFh     ;
1010201Ch/1013201Ch 2   Y2R_OFFSET_G  ; signed     R/W: FFFFh     ;
1010201Eh/1013201Eh 2   Y2R_OFFSET_B  ;/           R/W: FFFFh     ;
```

```
10102020h/10132020h 2   Y2R_ALPHA  ;bit7 for 5551  R/W: 000000FFh ;
10102100h/10132100h 4   Y2R_DITHER0                R/W: 0000CCCCh ;
10102108h/10132108h 4   Y2R_DITHER1                R/W: 0000CCCCh ;
10102110h/10132110h 4   Y2R_DITHER2                R/W: 0000CCCCh ;
10102118h/10132118h 4   Y2R_DITHER3                R/W: 0000CCCCh ;/
10302000h/10332000h 80h Y2R_INPUT_Y    ;aka Luma   W:             ;\
10302080h/10332080h 80h Y2R_INPUT_U    ;aka Cb     W:             ; FIFO's
10302100h/10332100h 80h Y2R_INPUT_V    ;aka Cr     W:             ; (ARM11)
10302180h/10332180h 80h Y2R_INPUT_YUYV ;Y1,U,Y2,V  W: (camera)    ;
10302200h/10332200h 80h Y2R_OUTPUT_RGBA            R: (RGBA)      ;/
```

10102000h/10132000h - Y2R_CNT (R/W)

```
0-2   Input Format YUV (0-4 = 422'8, 420'8, 422'16, 420'16, 422'BATCH)  (R/W)
3-7   Unused (0)
8-9   Output Format RGBA      (0=8888, 1=8880, 2=5551, 3=5650)          (R/W)
10-11 Output Clockwise Rotate (0=None, 1=90', 2=180', 3=270')           (R/W)
12    Output Swizzle          (0=LinearFramebuf, 1=MortonSwizzleTexture)(R/W)
13-14 Unused (0)
15    Unknown, reportedly used, but always 0 (maybe write-only?)          (?)
16    Brightness Dither Enable   (0=No, 1=Use Y2R_DITHER0-3)            (R/W)
17    Brightness Ugly Pulsation? (0=No, 1=Add 0,1,2,3 in frame 0,1,2,3) (R/W)
18-20 Unused (0)
21    Unknown, reportedly used, but always 0 (maybe write-only?)          (?)
22    Input DMA Enable               (0=Disable, 1=Enable CDMA 09h/15h) (R/W)
23    Output DMA Enable              (0=Disable, 1=Enable CDMA 0Ah/16h) (R/W)
24    Input DRQ Y?                   (0=No, 1=DRQ) (write 1 to ack?) (R/ack?)
25    Input DRQ U?                   (0=No, 1=DRQ) (write 1 to ack?) (R/ack?)
26    Input DRQ V?                   (0=No, 1=DRQ) (write 1 to ack?) (R/ack?)
27    Input DRQ YUYV (batch)         (0=No, 1=DRQ) (write 1 to ack?) (R/ack?)
28    Output DRQ RGB, 9th input line (0=No, 1=DRQ) (write 1 to ack?) (R/ack?)
29    Interrupt upon DRQ(s)?         (0=Disable, 1=Enable IRQ 4Bh/4Eh)  (R/W)
30    Interrupt upon Transfer done   (0=Disable, 1=Enable IRQ 4Bh/4Eh)  (R/W)
31    Start/Busy                     (0=Idle/Ready, 1=Start/Busy)       (R/W)
```

Input formats...

```
INPUT_YUV422_INDIV_8   0     is that 8bit? or 8x8pix? or divide by 8?
INPUT_YUV420_INDIV_8   1
INPUT_YUV422_INDIV_16  2     is that 16bit? or 16x16pix? or divide by 16?
INPUT_YUV420_INDIV_16  3
INPUT_YUV422_BATCH     4     aka camera YUYV
```

Conversion is done in units of 8 lines. However, the first Input block must be 9 lines (unless the total height is smaller), and the final blocks can be smaller (depending on amount of remaining lines). For example:

```
Send 9,8,8,8,8,3 scanlines    ;\for 44 scanlines
Recv 8,8,8,8,8,4 scanlines    ;/
```

The odd amount of sending 9 lines in 1st block might allow to keep converting data while receiving responses.

```
XXX does that 9-lines-input thing also apply to Input Format 0-3 ?
```

10102004h/10132004h - Y2R_WIDTH

```
0-2   Unused (0)
3-9   Width, in 8-pixel units (01h..7Fh=8..1016 pix, or 00h=?)
10-15 Unused (0)
```

10102006h/10132006h - Y2R_HEIGHT

```
3-9   Height in 1-pixel units (001h..3FFh=1..1023 pix, or 000h=?)
10-15 Unused (0)
```

10102008h/10132008h - Y2R_STROBE ;ack fifo? dummy r/w?

Unknown, there is no read/write-able data here, and the register is normally left unused, however, reading or writing-any-value seems to strobe/trigger/ack/reset some internal stuff?

Standard YUV to RGB Formula (aka YCbCr to RGB)

The standard formula uses the constants shown below. However, one can also different values in the

Y2R_MULTIPLIER_xxx and Y2R_OFFSET_xxx registers (that would allow to support other YUV variants, or to adjust more pale or more colorful output, or even to change RGB to BGR order; if the input FIFOs are also swapped accordingly).

```
R = Y*1.00 + (Cr-80h)*1.402
G = Y*1.00 - (Cr-80h)*0.714)-(Cb-80h)*0.344
B = Y*1.00 + (Cb-80h)*1.772
Clip results to MinMax(00h,FFh), and apply final divide by 8 for RGB555.
```

Note: In the standard YUV-to-RGB formula one would usually subtract 80h from the U/V values before multiplication. However, the 3DS does instead add variable offsets after multiplication (so those offsets need to be matched to the multipliers).

10102010h/10132010h - Y2R_MULTIPLIER_Y_TO_RGB (R/W) ;usually (1.000)*100h
10102012h/10132012h - Y2R_MULTIPLIER_V_TO_R (R/W) ;usually (1.402)*100h
10102014h/10132014h - Y2R_MULTIPLIER_V_TO_G (R/W) ;usually (0.714)*100h
10102016h/10132016h - Y2R_MULTIPLIER_U_TO_G (R/W) ;usually (0.344)*100h
10102018h/10132018h - Y2R_MULTIPLIER_U_TO_B (R/W) ;usually (1.772)*100h

```
0-9   Multiplier for the YUV-to-RGB formula (unsigned, 0..3FFh)
10-15 Unused (0)
```

1010201Ah/1013201Ah - Y2R_OFFSET_R (R/W) ;usually (-1.402)*1000h
1010201Ch/1013201Ch - Y2R_OFFSET_G (R/W) ;usually (+0.714+0.344)*1000h
1010201Eh/1013201Eh - Y2R_OFFSET_B (R/W) ;usually (-1.772)*1000h

```
0-15  Offset for the YUV-to-RGB formula (signed, -8000h..+7FFFh)
```

10102020h/10132020h - Y2R_ALPHA (R/W)

```
0-7   Alpha for RGBA output format 8888 and 5551 (the latter uses only bit7)
8-31  Unused (0)
```

10102100h/10132100h - Y2R_DITHER0 (R/W)
10102108h/10132108h - Y2R_DITHER1 (R/W)
10102110h/10132110h - Y2R_DITHER2 (R/W)
10102118h/10132118h - Y2R_DITHER3 (R/W)

```
0-31  Dither, R/W-mask 0000CCCCh
```

10302000h/10332000h - 80h byte window - Y2R_INPUT_Y (aka Luma) (W)
10302080h/10332080h - 80h byte window - Y2R_INPUT_U (aka Cb) (W)
10302100h/10332100h - 80h byte window - Y2R_INPUT_V (aka Cr) (W)
10302180h/10332180h - 80h byte window - Y2R_INPUT_YUYV (camera Y1,U,Y2,V) (W)
10302200h/10332200h - 80h byte window - Y2R_OUTPUT_RGBA (R)

```
0-31  FIFO data
```

The FIFOs are mirrored to 80h-byte windows, however, the transfer blocks with 8 or 9 scanlines are much longer than that (so better it's better to use a fixed FIFO address, instead of increasing addresses in the 80h-byte windows)

For the optional swizzling feature, see
3DS Video Texture Swizzling

3DS Video L2B Registers (RGB-to-RGBA Converter) (New3DS)

10130000h - New3DS - L2B_0 - First RGB-to-RGBA converter (New3DS only)
10131000h - New3DS - L2B_1 - Second RGB-to-RGBA converter (New3DS only)

```
10130000h/10131000h 4     L2B_CNT     Control  R/W  e3c00303h
10130004h/10131004h 2     L2B_WIDTH   Width    R/W  03f8h
10130006h/10131006h 2     L2B_HEIGHT  Height   R/W  03f8h
```

```
  10130020h/10131020h 4     L2B_ALPHA   Alpha    R/W  000000ffh
  10330000h/10331000h 1000h L2B_FIFO (IN and OUT, empty=data_abort) (R+W)
```
These registers appear to be called L2B, which might be short for Line-to-Block. They are similar to the YUV-to-RGBA converters, but merely converting from RGB-to-RGBA (and also allowing to convert between 5bit and 8bit color depths). And, they are always re-ordering the data from scanline format to swizzled texture format:
[3DS Video Texture Swizzling]

10130000h/10131000h - New3DS - L2B_CNT (R/W)
```
  0-1   Input RGBx Format            (0=8888, 1=8880, 2=5551, 3=5650)     (R/W)
  2-7   Unused (0)
  8-9   Output RGBA Format           (0=8888, 1=8880, 2=5551, 3=5650)     (R/W)
  10-21 Unused (0)
  22    Input DMA Enable             (0=Disable, 1=Enable CDMA 17h/19h)   (R/W)
  23    Output DMA Enable            (0=Disable, 1=Enable CDMA 18h/1Ah)   (R/W)
  24    Input DRQ                    (0=No, 1=DRQ) (write 1 to ack?)   (R/ack?)
  25    Output DRQ, 8th input line   (0=No, 1=DRQ) (write 1 to ack?)   (R/ack?)
  26-28 Unused (0)
  29    Interrupt upon DRQ(s)?       (0=Disable, 1=Enable IRQ 45h/46h)    (R/W)
  30    Interrupt upon Transfer done (0=Disable, 1=Enable IRQ 45h/46h)    (R/W)
  31    Start/Busy                   (0=Idle/Ready, 1=Start/Busy)         (R/W)
```
Send/Recv is done in units of 8 lines (unlike Y2R, which requires sending 9 lines in first block).

10130004h/10131004h - New3DS - L2B_WIDTH (R/W)
10130006h/10131006h - New3DS - L2B_HEIGHT (R/W)
```
  0-2   Unused (0)
  3-9   Width/Height in 8 pixel units (01h..7Fh=8..1016? pixels, or 00h=?)
  10-15 Unused (0)
```

10130020h/10131020h - New3DS - L2B_ALPHA (R/W)
```
  0-7   Alpha value for all pixels    (00h..FFh = Transparent..Solid)
  8-31  Unused (0)
```
Used as alpha for output format 8888 and 5551 (the latter uses only bit7 of the 8bit value).
Note: Any alpha values written to the input fifo are ignored, the alpha value is always taken from the alpha register, not from the incoming pixels.

10330000h/10331000h - New3DS - L2B_FIFO (R and W) (empty=data_abort)
```
  0-31  Pixel data
```
The FIFO is mirrored across a 1000h-byte window (but eight large scanlines may be larger than that, so it's better to used a fixed FIFO address instead of increasing addresses in that window).

3DS Video MVD Registers (Movie Decoder or so?) (New3DS)

Unknown. New3DS only. MVD might be short for Movie Decoder or so?
Unknown if there are any FIFOs and CDMA peripheral IDs... there are none known yet... maybe the thing does have it's own "DMA" hardware for direct memory access without needing CDMA?

10207000h - New3DS: Movie Decoder or so? Rockchip...?
```
  10207000h 4    R    67312398h  MVD Registers Chip ID?
  10207004h C4h  R/W  ffffffffh  MVD Registers
  102070C8h 4    R    07b4af80h  MVD Registers
  102070CCh 4    R/W  ffffffffh  MVD Registers
  102070D0h 4    -    00000000h  MVD Registers
  102070D4h 4    -    00000000h  MVD Registers
  102070D8h 4    R    c09a0000h  MVD Registers
  102070DCh 4    R/W  ffffffffh  MVD Registers
  102070E0h 4    -    00000000h  MVD Registers
```

```
102070E4h 4    R    8516ffffh  MVD Registers
102070E8h 4    -    00000000h  MVD Registers
102070ECh 40h  R/W  ffffffffh  MVD Registers
1020712Ch 4    -    00000000h  MVD Registers
10207130h 4    -    00000000h  MVD Registers
10207134h 4    -    00000000h  MVD Registers
10207138h 4    -    00000000h  MVD Registers
1020713Ch 44h  R/W  ffffffffh  MVD Registers
10207180h 4    -    00000000h  MVD Registers
10207184h 4    -    00000000h  MVD Registers
10207188h 4    -    00000000h  MVD Registers
1020718Ch 4    R    ffffffffh  MVD Registers
10207190h 4    R    ff874780h  MVD Registers
10207194h 6Ch  -    00000000h  Zerofilled
10207200h E00h Mirrors of above 200h byte area
```

The ID value 67312398h appears to be known as "HW_ID" for linux/android "VPU SERVICES". Searching for that two strings gives this source code,

```
http://git.jp.linux-rockchip.org/cgit/rk3288_r-box_android4.4.2_sdk/tree
/kernel/arch/arm/mach-rockchip/vcodec_service.c
```

which defines VPU_DEC_ID_9190=6731h (the upper 16bit of the 67312398h value), so the "MVD" might be identical to that Rockchip hardware (whatever that is).

3DS Video LGY Registers (Legacy GBA/NDS Video to Framebuffer)

The LGYFB units are for forwarding GBA/NDS/DSi video to 3DS screens with optional scaling. The input comes directly from the GBA/NDS video controller, the output must be DMAed to memory.
That is, ARM11 must handle that memory transfers in background while running GBA/NDS/DSi software on ARM7/ARM9 side.

10110000h - LGYFB_0 (Legacy Framebuffer 0) (NDS bottom screen) (and GBA)
10111000h - LGYFB_1 (Legacy Framebuffer 1) (NDS top screen) (and GBA)

```
10110000h/10111000h 4     LGYFB_CNT        R/W   mask:00019f37h  ;\
10110004h/10111004h 4     LGYFB_SIZE       R/W   mask:01ff01ffh  ; Control
10110008h/10111008h 4     LGYFB_IRQ_STAT   R/ack mask:01f80007h  ; Status
1011000Ch/1011100Ch 4     LGYFB_IRQ_ENABLE R/W   mask:0007h      ;/
10110020h/10111020h 4     LGYFB_ALPHA      R/W   mask:000000ffh  ;-Alpha
101100F0h/101110F0h 4     LGYFB_UNKNOWN    R/W   mask:0000000fh  ;-Unknown?
10110100h/10111100h 4     LGYFB_DITHER0    R/W   mask:0000cccch  ;\
10110108h/10111108h 4     LGYFB_DITHER1    R/W   mask:0000cccch  ; Dither
10110110h/10111110h 4     LGYFB_DITHER2    R/W   mask:0000cccch  ;
10110118h/10111118h 4     LGYFB_DITHER3    R/W   mask:0000cccch  ;/
10110200h/10111200h 4     LGYFB_V_LEN      R/W   mask:00000007h  ;\Vertical
10110204h/10111204h 4     LGYFB_V_PATTERN  R/W   mask:000000ffh  ; Scaling
10110240h/10111240h 4x30h LGYFB_V_ARRAY    R/W   mask:0000fff0h  ;/
10110300h/10111300h 4     LGYFB_H_LEN      R/W   mask:00000007h  ;\Horizontal
10110304h/10111304h 4     LGYFB_H_PATTERN  R/W   mask:000000ffh  ; Scaling
10110340h/10111340h 4x30h LGYFB_H_ARRAY    R/W   mask:0000fff0h  ;/
10310000h/10311000h 1000h LGYFB_FIFO       R     CDMA only       ;-Output
```

10110000h/10111000h - LGYFB_CNT (R/W)

```
0     Start/Enable               (0=Stop, 1=Start)                      (R/W)
1     Enable Vertical Scaling    (0=Disable, 1=Enable; via LGYFB_V_xxx) (R/W)
2     Enable Horizontal Scaling  (0=Disable, 1=Enable; via LGYFB_H_xxx) (R/W)
3     Unused (0)
4     Brightness Dither Enable   (0=No, 1=Use LGYFB_DITHER0-3)          (R/W)
5     Brightness Dither, too?    (as above, no Y2R-style Pulsation)     (R/W)
6-7   Unused (0)
8-9   Output Format RGBA      (0=8888, 1=8880, 2=5551, 3=5650)         (R/W)
10-11 Output Clockwise Rotate (0=None, 1=90', 2=180', 3=270')          (R/W)
12    Output Swizzle          (0=LinearFramebuf, 1=MortonSwizzleTexture)(R/W)
```

```
  13-14 Unused (0)
  15    Enable DMA              (0=Off, 1=Enable CDMA 0Dh/0Eh)            (R/W)
  16    Unknown... seems to have no visible effect for GBA/NDS (0=?, 1=?) (R/W)
  17-31 Unused (0)
```
Once when started, the transfer does auto-repeat each frame (although, that may hang with some/wrong settings; in that case it can help to toggle CNT.bit0 after DMAing the last block of each frame).

10110004h/10111004h - LGYFB_SIZE (R/W)
```
  0-8   Output Width (after scaling), minus 1  (0..1FFh = 1..512 pixels)  (R/W)
  9-15  Unused (0)
  16-24 Output Height (after scaling), minus 1 (0..1FFh = 1..512 pixels)  (R/W)
  25-31 Unused (0)
```
Caution: Must be written via 32bit STR (trying to use 16bit STRH will set BOTH halfwords to the same value).

10110008h/10111008h - LGYFB_IRQ_STAT (R/ack)
```
  0     First 8-Line Output Block (0=No, 1=Yes/IRQ) (write 1 to clear)  (R/ack)
  1     Next 8-Line Output Block  (0=No, 1=Yes/IRQ) (write 1 to clear)  (R/ack)
  2     Last Input? Line          (0=No, 1=Yes/IRQ) (write 1 to clear)  (R/ack)
  3-15  Unused (0)
  16-24 Output Block Line Number for IRQ bit0/1 (step 8)                    (R)
  25-31 Unused (0)
```
The Output Block Line Number can be used to compute the destination address for IRQ bit0/1. The initial line number upon reset is random/garbage (but gets valid after setting LGYFB_CNT.bit0).
Overrun can occur when not reading the output FIFO fast enough. After overrun, bit1 triggers only on each 2nd block, and bit2 won't trigger at all.

1011000Ch/1011100Ch - LGYFB_IRQ_ENABLE - ? (R/W)
```
  0     First 8-Line Output Block (0=Off, 1=Enable IRQ 4Ch/4Dh)           (R/W)
  1     8-Line Output Blocks      (0=Off, 1=Enable IRQ 4Ch/4Dh)           (R/W)
  2     Last Input? Line          (0=Off, 1=Enable IRQ 4Ch/4Dh?)          (R/W)
  3-31  Unused (0)
```
IRQ enable does also require LGYFB_CNT.bit0=1 and CFG11_TWLMODE_BOOT.bit15=1.
The end of frame irq occurs only if the blocks were actually transferred (via CDMA).

10110020h/10111020h - LGYFB_ALPHA - (R/W)
```
  0-7   Alpha value for all pixels    (00h..FFh = Transparent..Solid)
  8-31  Unused (0)
```
Used as alpha for output format 8888 and 5551 (the latter uses only bit7 of the 8bit value).

101100F0h/101110F0h - LGYFB_UNKNOWN (R/W)
```
  0-3   Unknown (initially 0Fh on reset)
  4-31  Unused (0)
```
Unknown. IRQ and DMA requests won't occur when using too small values. Without scaling values 01h..0Fh are working, with 2x vertical scaling only values 06h..0Fh are working.

10110100h/10111100h - LGYFB_DITHER0 (R/W)
10110108h/10111108h - LGYFB_DITHER1 (R/W)
10110110h/10111110h - LGYFB_DITHER2 (R/W)
10110118h/10111118h - LGYFB_DITHER3 (R/W)
```
  0-31  Dither alike Y2R, R/W-mask 0000CCCCh
```

10310000h/10311000h - LGYFB_FIFO (R)
Caution: This FIFO works via CDMA only (unlike most or all other FIFOs, it does trigger data abort when trying to read via CPU LDR opcodes; even if there is data in the FIFO).
```
  0-31  Output FIFO (contains 8 output lines per DMA request)
```
Use DMAWFP opcode (Wait for Peripheral) before reading an 8-line block. Or, wait for LGYFB_IRQ_STAT bit0/1, and then manually start the DMA for one 8-line block (the latter can be useful for transfers with clockwise rotate; where one may need to patch the destination address for each block).

For the optional swizzling feature, see
[3DS Video Texture Swizzling]

```
_____________________________ Scaling Unit _____________________________
```

10110200h/10111200h - LGYFB_V_LEN - Vertical scaling (R/W)
10110300h/10111300h - LGYFB_H_LEN - Horizontal scaling (R/W)

```
  0-2   Batch size-1  (0..7 = 1..8 dst pixels) (using (1..8)*6 array entries)
  3-31  Unused (0)
```

Selects the number of pattern bits and array entries to be used (before repeating the scaling pattern).

10110204h/10111204h - LGYFB_V_PATTERN - Vertical scaling (R/W)
10110304h/10111304h - LGYFB_H_PATTERN - Horizontal scaling (R/W)

```
  0-7   Read a new src pixel before computing 1st..8th dst pixel (0=No, 1=Yes)
  8-31  Unused (0)
```

"The amount of set bits determine how many pixels are read each batch."
"Any bit indexes past LGYFB_x_LEN are ignored."
"This value is 8 bits, but it has to be written with a 32bit write."
Example values:

```
  Len Pattern    Effect
  1   xxxxxxx1b  No scaling    (1 input pixels --> 1 output pixels)
  8   11111111b  No scaling    (8 input pixels --> 8 output pixels)
  5   xxx01111b  Scale by 1.25 (4 input pixels --> 5 output pixels) ;NDS/DSi
  4   xxxx0111b  Scale by 1.33 (3 input pixels --> 4 output pixels)
  3   xxxxx011b  Scale by 1.5  (2 input pixels --> 3 output pixels)
  6   xx011011b  Scale by 1.5  (4 input pixels --> 6 output pixels) ;GBA
  3   xxxxx001b  Scale by 3    (1 input pixels --> 3 output pixels)
  2   xxxxxx01b  Scale by 2    (1 input pixels --> 2 output pixels)
  8   01010101b  Scale by 2    (4 input pixels --> 8 output pixels)
 GBA (240x160) scale by 1.5  = 3DS top screen (360x240)
 GBA (240x160) scale by 1.33 = 3DS bottom screen (320x213)
 NDS (256x192) scale by 1.25 = 3DS either screen (320x240)
```

10110240h/10111240h - LGYFB_V_ARRAY - Vertical scaling, 6x8 words (R/W)
10110340h/10111340h - LGYFB_H_ARRAY - Horizontal scaling, 6x8 words (R/W)

This array contains 6x8 words, used to compute up to 8 output pixels, with brightness multipliers for 6 input pixels each.

```
  0-3   Unused (0) (Nintendo writes 16bit to bit0-15, but bit0-3 are ignored)
  4-15  Brightness per source pixel (signed, -800h..+7FFh; 400h=full/max)
  16-31 Unused (0)
```

The sum of six input values should be 400h (the hardware does automatically clip results to min/max brightness; clipping can happen when mixing positive and negative values; with some of them getting multiplied with dark input pixels).
Note: Multipliers bigger than 400h are glitchy (value 7FFh somehow converts white pixels to dark gray).

Default Array for GBA screen (scale by 1.5) (240x160 to 360x240)

This is using Pattern=011011b and Length=6 (minus 1). The array entries are straight ahead, using pixels with full brightness, and merged pixels with half brightness:

```
  0000h,0000h,0000h,0000h,0000h,0000h, N/A , N/A  <-- for 1st input pixel
  0000h,0000h,0000h,0000h,0000h,0000h, N/A , N/A  <-- for 2nd input pixel
  0000h,2000h,4000h,0000h,2000h,4000h, N/A , N/A  <-- for 3rd input pixel
  4000h,2000h,0000h,4000h,2000h,0000h, N/A , N/A  <-- for 4th input pixel
  0000h,0000h,0000h,0000h,0000h,0000h, N/A , N/A  <-- for 5th input pixel
  0000h,0000h,0000h,0000h,0000h,0000h, N/A , N/A  <-- for 6th input pixel
    |     |     |     |     |     |
    |                          '-----------------> to 6th output pixel
    '----------------------------------------------> to 1st output pixel
```

For whatever reason, this is scaling by 6:4 with 6 output pixels (instead of 3:2 with 3 output pixels). Unknown if it's faster that way, or if there's some other advantage.

Default Array for NDS/DSi screens (scale by 1.25) (256x192 to 320x240)
This is using Pattern=01111b and Length=5 (minus 1). The array entries contain positive and negative values, which might raise contrast between bright/dark pixels:

```
0000h,004Eh,011Dh,01E3h,01C1h, N/A , N/A , N/A  <-- for 1st input pixel
0000h,FCA5h,F8D0h,F69Dh,F873h, N/A , N/A , N/A  <-- for 2nd input pixel
0000h,0D47h,1E35h,2F08h,3B6Fh, N/A , N/A , N/A  <-- for 3rd input pixel
4000h,3B6Fh,2F08h,1E35h,0D47h, N/A , N/A , N/A  <-- for 4th input pixel
0000h,F873h,F69Dh,F8D0h,FCA5h, N/A , N/A , N/A  <-- for 5th input pixel
0000h,01C1h,01E3h,011Dh,004Eh, N/A , N/A , N/A  <-- for 6th input pixel
  |     |     |     |     |
  |                       '-----------------------> to 5th output pixel
  '-------------------------------------------------> to 1st output pixel
```

Weirdly, the values for 2nd-5th output pixel values sum up to 3FDDh/3FAAh (actually less, because the lower 4bit are ignored), making them a bit darker than 1st output pixel.

3DS Video Texture Swizzling

Texture Swizzling
Morton Swizzling, or Z-order Swizzling is done by reading source pixels from a scanline based bitmap in "Z-shaped" read-direction, and then storing that pixels at continous VRAM addresses.
In the drawing below, each "Z" represents 2x2 pixels (arranged as a "Z" shape, ie. upper-left, upper-right, lower-left, lower-right). On a larger scale, each 2x2 Z's are also forming a larger Z, and so on.

```
Z/Z  /Z/Z
.-' / .-'
Z/Z/  Z/Z
    ..--'
.--'
Z/Z  /Z/Z
.-' / .-'
Z/Z/  Z/Z
```

This can improve cache hits for adjacent pixels. In a large bitmap, pixels in adjacent scanlines are always located in separate cache entries. With the swizzling, there is a better chance that they are in the same cache entry.

Addressing
The swizzling does interleave the x/y address bits. On the 3DS hardware, data is processed in units of 8 scanlines, so the interleave occurs on lower three x/y bits only. For example, for 256x256 pixel data:

```
Scanline-based bitmap -->  YyyyyyyyXxxxxxxx
Swizzled texture      -->  YyyyyXxxxxyxyxyx
```

Swizzling Hardware
The 3DS has some hardware for converting scanlines to swizzled textures:

```
Y2R Registers (YUV-to-RGBA)
L2B Registers (RGB-to-RGBA) (New3DS only)
LGYFB Registers (GBA/NDS/DSi-to-3DS-Framebuffer)
```

Swizzling Examples
Examples for scanline pixels before/after swizzling:

```
Linear scanlines, 16x8 pixels:                   Linear lines, 8x8 pixels:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   00 01 02 03 04 05 06 07
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F   08 09 0A 0B 0C 0D 0E 0F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   10 11 12 13 14 15 16 17
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F   18 19 1A 1B 1C 1D 1E 1F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F   20 21 22 23 24 25 26 27
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F   28 29 2A 2B 2C 2D 2E 2F
```

```
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F   30 31 32 33 34 35 36 37
70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F   38 39 3A 3B 3C 3D 3E 3F
Swizzled texture, 2 tiles:                        Swizzled texture, 1 tile:
00 01 10 11 02 03 12 13  ;\                       00 01 08 09 02 03 0A 0B
20 21 30 31 22 23 32 33  ;                        10 11 18 19 12 13 1A 1B
04 05 14 15 06 07 16 17  ;                        04 05 0C 0D 06 07 0E 0F
24 25 34 35 26 27 36 37  ; left tile              14 15 1C 1D 16 17 1E 1F
40 41 50 51 42 43 52 53  ;                        20 21 28 29 22 23 2A 2B
60 61 70 71 62 63 72 73  :                        30 31 38 39 32 33 3A 3B
44 45 54 55 46 47 56 57  ;                        24 25 2C 2D 26 27 2E 2F
64 65 74 75 66 67 76 77  ;/                       34 35 3C 3D 36 37 3E 3F
08 09 18 19 0A 0B 1A 1B  ;\
28 29 38 39 2A 2B 3A 3B  ;
0C 0D 1C 1D 0E 0F 1E 1F  ;
2C 2D 3C 3D 2E 2F 3E 3F  ; right tile
48 49 58 59 4A 4B 5A 5B  ;
68 69 78 79 6A 6B 7A 7B  :
4C 4D 5C 5D 4E 4F 5E 5F  ;
6C 6D 7C 7D 6E 6F 7E 7F  ;/
```

Note: Z-order refers to the shape of the letter "Z", not to be confused with Z-axis, and not to be confused with the zigzag-order that is used in JPEGs.

3DS Sound and Microphone

Unknown how to output sound. In first place, it may require a console with intact top screen (with speakers, and volume slider) (headphones might work without that, provided that absent slider defaults to "max" volume, or that one could perhaps override the slider via MCU/BPTWL/TSC/etc).
Moreover, one will probably need to init TSC and SNDEXCNT, and possibly CFG11 and GPIO and whatever other registers.
The capture unit works without top screen, so some testing is already possible.

_____ DSP Registers _____

Reportedly 10141220h - CFG11_CODEC_CNT also has a DSP related bit?

10203000h Teak DSP Registers
These registers are disabled by default (always zero), use CFG11_DSP_CNT (Port 10141230h) to enable them.

```
10203000h 2    DSP_PDATA
10203004h 2    DSP_PADR
10203008h 2    DSP_PCFG
1020300Ch 2    DSP_PSTS
10203010h 2    DSP_PSEM
10203014h 2    DSP_PMASK
10203018h 2    DSP_PCLEAR
1020301Ch 2    DSP_SEM
10203020h 2    DSP_CMD0
10203024h 2    DSP_REP0
10203028h 2    DSP_CMD1
1020302Ch 2    DSP_REP1
10203030h 2    DSP_CMD2
10203034h 2    DSP_REP2
10203040h FC0h Mirrors of above 40h-byte area
10203200h 40h  Reportedly "LCD", but actually it's just one of above mirrors
```

These are same as on DSi.
DSi XpertTeak (DSP)
3DS software is commonly using the DSP to play music recordings in AAC format, and reportedly there is

something called DSP-ADPCM (maybe something similar to IMA-ADPCM)?

CSND Registers

The sound hardware works very similar to NDS sound, see there for details about PCM, IMA-ADPCM and PSG formats.
DS Sound
Differences are twice as many sound channels, some moved control bits, and simplified volume control (without sound/panning/master volumes).
Unknown if/where BIAS exists on 3DS?

10103000h - Sound Control (R/W)

```
0-15   CSND Master Volume (0..8000h) (8001h..FFFFh=replaced by 8000h) (R/W)
16     Disable sound output           (0=Enable, 1=Mute)              (R/W)
17-29  Unused? (0)
30     Dissonant sound??              (0=Dissonant, 1=Normal)         (R/W)
31     Makes some register/bits R/W   (0=No, 1=Yes)                   (R/W)
```

When bit31=1:

```
10103400h+(N*20h).bit15   becomes R/W
1010340Ch+(N*20h).bit31-2 becomes R/W
10103800h+(N*10h).bit15   becomes R/W
1010380Ch+(N*10h).bit31-0 becomes R/W
```

CSND Channel Registers

10103400h+(N*20h) - Sound Channel 0-31: Control (parts W and R/W)

```
0-2    Wave Duty   (0..7) ;HIGH=(N+1)*12.5%, LOW=(7-N)*12.5% (PSG only) (R/W)
3-5    Unused (0)
6      Linear interpolation on each two samples (0=Disable, 1=Enable)   (R/W)
7      Hold   (0=Normal, 1=Hold last sample after one-shot sound)       (R/W)
8-9    Unused (0)
10-11  Repeat (0=Manual, 1=Loop Infinite, 2=One-shot, 3=Same as 1?)     (R/W)
12-13  Format (0=PCM8, 1=PCM16, 2=IMA-ADPCM, 3=PSG/Noise)               (R/W)
14     Pause DMA or so? (0=Pause?, 1=Normal)  ;no effect on PSG?!       (R/W)
15     Start/Status (0=Stop, 1=Start/Busy)    ;need 10103000h.bit31     (R/W)
16-31  Sample Rate (0..FFBEh; 0=Slowest, FFBEh=Fastest) (FFBFh..=Hangs)   (W)
```

PSG/Noise: Rectangular waves are supported on channel 8-13, noise on channel 14-15. Rectangular wave Duty is same as for NDS (but 32 samples long instead of 8 samples).
Sample rate is reportedly 4x higher than NDS:

```
value = 67.027964MHz / samplerate
uh, that value is NOT negative??? apparently wrong.
XXX but what is NOISE frequency?
```

10103404h+(N*20h) - Sound Channel 0-31: Output Volume (R/W)

```
0-15  Volume Right (0..8000h)  ;\writing values 8001h..FFFFh is
16-31 Volume Left  (0..8000h)  ;/automatically changed to 8000h
```

10103408h+(N*20h) - Sound Channel 0-31: Capture Volume (R/W)

```
0-15  Capture 0 volume (right?) (0..8000h) ;\writing 8001h..FFFFh gets
16-31 Capture 1 volume (left?)  (0..8000h) ;/automatically changed to 8000h
```

1010340Ch+(N*20h) - Sound Channel 0-31: Start Address (R/W) (Bit0-1: W)
10103414h+(N*20h) - Sound Channel 0-31: Loop Restart Address (W)

```
0-31  Physical Memory Address (unused for PSG/noise)
```

The R/W-ability is a mess: Start.bit2-31 are R/W (if enabled in 10103000h.bit31). Start.bit0-1 do also exist, but are write-only. Restart.bit0-31 are write-only.

10103410h+(N*20h) - Sound Channel 0-31: Total Size (W)

```
0-26   Size in bytes (0..7FFFFFFh) (unused in manual mode)
```

```
  27-31  Unknown/Unused (0)
```
The size value is for the total size from Start to End (or Loop End if looped). The total size (and size of looped part) semm to be required to be at least 10h. Hardware behaves quirky if the size of the looped part (total-(restart-start)) is odd and/or(?) less than 10h. And, the quirkyness can differ depending on whether CNT.bit10-11 is 1 or 3 (which is otherwise same).

10103418h+(N*20h) - Sound Channel 0-31: Start IMA-ADPCM state (W)
```
  0-15   Initial PCM16 Value (Pcm16bit = -7FFFh..+7FFF) (not -8000h)
  16-22  Initial Table Index Value (Index = 0..88)
  23     Uh, reportedly MSB of above 7bit value ???
  ..     Unknown/Unused (0)
```
Equivalent to the IMA-ADPCM header from NDS samples (but, unlike as NDS, the header must be manually written here; instead being located in the first four sample bytes).

1010341Ch+(N*20h) - Sound Channel 0-31: Loop Restart IMA-ADPCM state (W)
```
  0-15   Restart PCM16 Value (Pcm16bit = -7FFFh..+7FFF) (not -8000h)
  16-22  Restart Table Index Value (Index = 0..88)
  23     Uh, reportedly MSB of above 7bit value ???
  ..     Unknown/Unused (0)
  31     Reportedly ADPCM state reload at Loop Restart (1=Enable)
```
Allows to force the ADPCM state to the correct (pre-calculated) value upon looping (or, with bit31=0, samples could drift if the last sample doesn't end with exact same state as the first loop sample)...?
Uh, that's both weird... on NDS, the hardware did simply latch the correct value automatically... is that really left unsupported on 3DS?

_____ CSND Capture Registers _____

These are probably for left/right channel...
Maybe Capture 0 Left, and Capture 1 right?

10103800h+(N*10h) - Sound Capture 0-1: Control (R/W)
```
  0      Capture Repeat        (0=Loop, 1=One-shot)
  1      Capture Format        (0=PCM16, 1=PCM8)
  2      Unknown ...? maybe something similar as on NDS
  2-14   Unused (0)
  15     Capture Start/Status  (0=Stop, 1=Start/Busy)  ;need 10103000h.bit31
  16-31  Unused (0)
```
There seems to be some extra delay at begin or end of capture: For one-shot, the bit15=1 duration should depend on rate and length (but takes a bit longer than it should).

10103804h+(N*10h) - Sound Capture 0-1: Sample Rate (W)
```
  0-15   Sample Rate (0..FFBEh; 0=Slowest, FFBEh=Fastest) (FFBFh..FFFFh=Hangs)
  16-31  Unknown/unused (0)
```

1010380Ch+(N*10h) - Sound Capture 0-1: Length (W)
```
  0-23   Length (in bytes) (1..FFFFFFh) (and oddly: 0=Same as 4)
  24-31  Unknown/unused (0)
```

1010380Ch+(N*10h) - Sound Capture 0-1: Address (R/W)
```
  0-31   Address (in bytes) <--- This is R/W only if enabled in 10103000h.bit31
```
Capture works even if address and size are unaligned (even so for PCM16).

_____ CODEC Registers _____

10145000h - CODEC_SNDEXCNT, reportedly "Empty" uh? (R/W)
```
  0-5   DSP sound volume (00h..3Fh) (R/W)
  6-11  GBA sound volume (00h..3Fh) (R/W)
  12    Enable Microphone timing
```

```
  13    Sound/Microphone I2S frequency (0=32.73 kHz, 1=47.61 kHz)
  14    Unknown (maybe Mute, alike DSi's SNDEXCNT)
  15    Enable Microphone and Teak BTDMP timing and GBA sound
  16-28 Unused (0)
  29    Unknown                   (R/W)
  30    Unknown, disonnant sound? (R/W)
  31    Master Enable Microphone (and Sound?)
```
Microphone requires bit31 set, and also one or both of bit15/bit12 set.
Teak BTDMP timing requires bit15 set (and is also affected by bit13).
GBA sound requires bit15 set, and nonzero volume in bit6-11.
Maybe some further bits here resemble DSi's SNDEXCNT...?

```
_____________________________ MIC Registers _____________________________
```

MIC Registers
```
  10162000h 2  REG_MIC_CNT  Microphone Control
  10162004h 4  REG_MIC_DATA Microphone Data
```
These are about same as for DSi.
[DSi Microphone and SoundExt]
The microphone must be unmuted in touchscreen TSC registers.
The microphone must be enabled in Port 10145000h.
Microphone triggers CDMA 00h (requires Port 1014010Ch, CFG11_CDMA_CNT.bit0=1).

3DS Cartridge Registers

Gamecard related CONFIG9 Registers
```
  1000000Ch 2   CFG9_CARD_CTL           NTRCARD/CTRCARD/SPI controller select
  10000010h 1   CFG9_CARD_POWER         Power On/Off/Reset and Eject detect
  10000012h 2   CFG9_CARD_INSERT_DELAY  Cart insert detection delay
  10000014h 2   CFG9_CARD_PWROFF_DELAY  Cart power off delay
```
[3DS Config - CONFIG9 Registers]

```
_____________________________ 3DS Cartridge Protocol _____________________________
```

Cartridge detection is done using unencrypted 8-byte NTRCARD commands, 3DS cartridges have ChipID.bit28=1, if so, command 3Eh is used to switch to 16-byte CTRCARD mode.
Of these 16-commands, the first command/reply (82h) is unencrypted, used to receive the header with an AES-encrypted encryption constant. The next command/reply (83h) is encrypted using that constant, used to send a random value. The following commands are encrypted using a combination of the original constant and the random value.
```
  Command                          Data  Expl.
  9F00000000000000                 2000h Dummy
  71C93FE9BB0A3B18                 0     Unknown/dummy (hardcoded constant)
  9000000000000000                 4     Get Chip ID, response=9000FEC2
  9000000000000000                 4     Get Chip ID, response=9000FEC2
  A000000000000000                 4     Get Chip Type, response=00000000
  3E00000000000000                 0     Enter 16-byte command mode
  82000000000000000000000000000000 200h  Get header/seed (NCSD[1000h..11FFh])
  83000000000000000rrrrrrrrrrrrrrr 0     Change Seed (to random)
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  A300000000000000rrrrrrrrrrrrrrrr 4     Get Chip Type, response=00000000
  C500000000000000rrrrrrrrrrrrrrrr 0     Unknown Watchdog?
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  A200000000000000rrrrrrrrrrrrrrrr 4     Get Chip ID, response=9000FEC2
  BF00000000000000000000000000000000 200h  Read address 0 (NCSD[0..1FFh])
```

```
BF00000000004000000000000000000 200h  Read address 4000h (maybe partition?)
C6000000000000000000000000000000 40h   Get Unique ID (on-chip PROM)
BFaaaaaaaaaaaaaa0000000000000000 200h  Read address aaa...
C5000000000000000000000000000000 0     Unknown Watchdog (each 10,000 reads)
BFaaaaaaaaaaaaaa0000000000000000 200h  Read address aaa...
```

On a lower level, the NTRCARD commands are working about as on NDS/DSi (with hardcoded gaps between command and data). The CTRCARD commands are sending a status byte instead of gaps (00h bytes when not ready, 01h when ready for data transfer; the 3DS hardware does automatically wait for that status byte before transferring data blocks). The data block is reportedly followed by a standard CRC32 value across unencrypted data. However, cartridges do only seem to output correct CRC's for command 82h/BFh. And, even then, the CRC doesn't actually look like a standard CRC32, or maybe the CRC gets garbled by encryption or status bytes?

The PROM is used for online games, it contains a 16-byte hex number (followed by 30h-byte FFh-filled area). Some card do also have writeable NAND instead of ROM. The NAND is also accessed via CTRCARD0 registers (not CTRCARD1), the NAND commands are still unknown.

3DS Cartridge Encryption

The encryption seed is initialized as follows:

```
CardType = (ReplyFromCommandA0h AND 00000003h)
send CTRCARD command 82h                    ;-receive NCSD[1000h..11FFh]
AES.KEYX = as set by bootrom, keyslot 3Bh   ;\
AES.KEYY = NCSD[1000h..100Fh]               ; decrypt seed via AES-CCM
AES.MAC  = NCSD[1020h..102Fh]               ; (use big-endian input,
AES.IV   = NCSD[1030h..103Bh]               ; and little-endian output)
if CardType=3 then AES.KEY = zerofilled     ;  <--dev card (keyslot 11h)
AES.DATA.IN  = NCSD[1010h..101Fh]           ;
ctrcard_seed[00h..0Fh] = AES.DATA.OUT       ;/
CTRCARD_SECSEED = ctrcard_seed              ;\
CTRCARD_SECCNT = CardType*100h+8004h        ; apply fixed seed
wait until CTRCARD_SECCNT.bit14=1           ;/
random64bit = whatever, can be zero, or same as fixed seed, or random
   cmdRand1 = REG_PRNG[0];
   cmdRand2 = REG_PRNG[4];
send CTRCARD command 83h + random64bit      ;-send random64bit to card
ctrcard_seed[00h..07h] = random64bit        ;-change LSBs of fixed seed
CTRCARD_SECSEED = ctrcard_seed              ;\
CTRCARD_SECCNT = CardType*100h+8004h        ; apply random seed
wait until CTRCARD_SECCNT.bit14=1           ;/
send further CTRCARD commands...            ;-get chip id, read data, etc.
```

The inner workings of the encryption hardware are unknown, it does probably contain AES hardware with some built-in secret key/iv values. Cart dumping is possible using the 3DS console hardware, there are also "sky3DS" flashcard that can run pirate copies on 3DS consoles without needing exploits, but that's kinda useless and works only for RSA signed official titles with region lock (the game selection is done via buttons on the cartridge instead of file menu).

NTRCARD Registers

NTRCARD Registers (ARM9/ARM11)

These registers are same as the old NDS Cartridge registers, see
DS Cartridge I/O Ports

```
10164000h 2  REG_NTRCARD_MCNT    ;40001A0h 2  Gamecard ROM and SPI Control
10164002h 2  REG_NTRCARD_MDATA   ;40001A2h 2  Gamecard SPI Bus Data/Strobe
10164004h 4  REG_NTRCARD_ROMCNT  ;40001A4h 4  Gamecard bus timing/control
10164008h 8  REG_NTRCARD_CMD     ;40001A8h 8  Gamecard bus 8-byte command out
10164010h 4  REG_NTRCARD_SEEDX_L ;40001B0h 4  Gamecard Encryption Seed 0 Low
10164014h 4  REG_NTRCARD_SEEDY_L ;40001B4h 4  Gamecard Encryption Seed 1 Low
10164018h 1  REG_NTRCARD_SEEDX_H ;40001B8h 2  Gamecard Encryption Seed 0 High
1016401Ah 1  REG_NTRCARD_SEEDY_H ;40001BAh 2  Gamecard Encryption Seed 1 High
1016401Ch 4  REG_NTRCARD_FIFO    ;4100010h 4  Gamecard bus 4-byte Data In (R)
```

These registers are usually used on ARM9 side. However, they are mapped to both ARM11/ARM9, and do support IRQ/DMA on ARM11/ARM9 side (unlike most other registers in that area, which support IRQ/DMA on ARM11 side only).
Note: ARM11-side CDMA 01h requires Port 1014010Ch, CFG11_CDMA_CNT.bit1=1.

_____ CTRCARD Registers _____

CTRCARD Registers (ARM9)

```
10004000h - CTRCARD0 - normal cartridge access
10005000h - CTRCARD1 - unknown purpose, not used for ROM cards, nor NAND cards
 10004000h/10005000h 4   CTRCARD_CNT
 10004004h/10005004h 4   CTRCARD_BLKCNT
 10004008h/10005008h 4   CTRCARD_SECCNT    ;varies for CARD0/CARD1
 1000400Ch/1000500Ch 4   CTRCARD_LOCK
 10004010h/10005010h 4   CTRCARD_SECSEED      FIFO!!!!!
 10004020h/10005020h 16  CTRCARD_CMD
 10004030h/10005030h 4   CTRCARD_FIFO
```

10004000h/10005000h - CTRCARD_CNT

```
 0-4   Timeout (0-16=1ms,2ms,4ms,8ms,..,64s; 17-31=64s, too; def=12=4s)  (R/W)
 5     Timeout Error      (0=Okay, 1=Error) (write 0 to ack)           (R/ack)
 6     Timeout Enable     (0=Disable, 1=Enable)                          (R/W)
 7     Unused (0)
 8     CRC Error          (0=Okay, 1=Error) (write 0 to ack)           (R/ack)
 9     CRC Enable         (0=Disable, 1=Enable) (works for cmd 82h/BFh)  (R/W)
 10-14 Unused (0)
 15    DMA Enable         (0=Disable, 1=Enable DMA DRQs, each 8 words)   (R/W)
 16-19 Data Block size    (0-8=0,4,16,64,512,1K,2K,4K,8K; 9-15=8K, too)  (R/W)
 20-23 Unused (0)
 24-26 Transfer Clock     (0-5=67MHz div 4,5,6,8,10,16; 6-7=div16, too)  (R/W)
 27    Data-Word status   (0=Busy, 1=Ready/DRQ)                            (R)
 28    Reset Pin          (0=Low/Reset, 1=High/Release) (SET-ONCE)       (R/W)
 29    Transfer Direction (0=Read, 1=Write)                              (R/W)
 30    Interrupt Enable   (0=Disable, 1=Enable) (ARM9 IF.bit23/24)       (R/W)
 31    Start              (0=Idle, 1=Start/Busy)                         (R/W)
```

With the above Transfer Clock settings, the clock (per byte) can range from 16.7MHz downto 4.2MHz.
Once reset pin is set high, it cannot be changed until controller is reset (which can be done CFG9_CARD_POWER). Setting bit28 does work only when writing bit31=0.

10004004h/10005004h - CTRCARD_BLKCNT (R or R/W)

```
 0-14  Number of data blocks to read, minus 1   (0..7FFFh=1..8000h)      (R/W)
 15    Unused (0)
 16-28 Number of data blocks to write, minus 1  (0..1FFFh=1..2000h)      (R/W)
 29-31 Unused (0)
```

10004008h/10005008h - CTRCARD_SECCNT (R or R/W)

For CARD0 (10004008h): (R/W mask 0307h, with readonly mask 4000h)

```
 0-1   Crypto Mode   (0=Normal, 1=Unknown, 2=Ignore SEED, 3=same as 0)   (R/W)
 2     Crypto Enable (0=Disable, 1=Enable)                               (R/W)
 3-7   Unused (0)
 8-9   Crypto Key index      (0..3, from A0h command) (3=debug)          (R/W)
 10-14 Unused (0)
 14    Crypto Apply Ready    (0=Busy, 1=Ready)                             (R)
 15    Crypto Apply Seed/Key (0=No, 1=Update Seed, works only if bit2=1)   (W)
 16-31 Unused (0)
```

For CARD1 (10005008h): (R/W mask 9300h, without readonly mask)

```
 0-7   Unused (0)
 8-9   Key index (?)                                                     (R/W)
 10-11 Unused (0)
 12    unknown... ?                                                      (R/W)
 13-14 Unused (0)
```

```
15    unknown... ?                                                    (R/W)
16-31 Unused (0)
```

1000400Ch/1000500Ch - CTRCARD_LOCK (R or R/W)

```
0     Write-protect CNT.bit28, SECCNT.bit0-2 (0=No, 1=Lock) (SET-ONCE)  (R/W)
1-31  Unused (0)
```

10004010h/10005010h - CTRCARD_SECSEED (W)

```
0-31  Encryption Seed, 4-word FIFO (16 bytes)
```

To apply the four most recent words written to SECSEED, write SECCNT.bit15=1, then wait for SECCNT.bit14=1.
The seed is usually updated twice: Once after command 82h (using four seed words from the cart header), and once after command 83h (with two seed words changed to random values).

10004020h/10005020h - CTRCARD_CMD (W)

```
0-127 Command (128bit, aka 16 bytes, little endian, transferred MSB first)
```

10004030h/10005030h - CTRCARD_FIFO (R) (maybe also for write-direction?)

```
0-31  Data (from 8 word FIFO)
```

```
_____________________________ SPI CARD Registers _____________________________
```

SPI FLASH cartridge savedata is accessed via SPI_CARD registers, see:
3DS SPI Registers

3DS Interrupts and Timers

ARM9
3DS ARM9 Interrupts
3DS ARM9 Timers

ARM11
3DS ARM11 Interrupts
ARM11 MPCore Private Memory Region Register Summary
ARM11 MPCore - Snoop Control Unit (SCU)
ARM11 MPCore - Timer and Watchdog
ARM11 MPCore - Interrupt Configuration
ARM11 MPCore - Interrupt Handling
ARM11 MPCore Distributed Interrupt Controller (Blurb)

3DS ARM9 Interrupts

ARM9 Interrupts
Mostly same as GBA/NDS/DSi, but without IME (only CPSR irq enable), and without IE2/IF2, and with changed IRQ bits:
10001000h - IRQ_IE - ARM9 Interrupt Enable
10001004h - IRQ_IF - ARM9 Interrupt Flags

```
0-7   NDMA 0..7                                              (Port 10002000h)
8-11  TIMER 0..3                                             (Port 10003000h)
12    PXI_SYNC        ;aka IPC Sync                          (Port 10008000h)
13    PXI_NOT_FULL    ;aka IPC Send FIFO Empty      ?        (Port 10008000h)
14    PXI_NOT_EMPTY   ;aka IPC Recv FIFO Not Empty  ?        (Port 10008000h)
15    AES                                                    (Port 10009000h)
```

```
16    SDMMC controller (eMMC and SD/MMC slot)             (Port 10006000h)
17    SDMMC sdio.irq pin?
18    SDxx controller?    ;maybe this is SDIO wifi ? or unused 2nd SDMMC?
19    SDxx sdio.irq pin?
20    DEBUG_RECV      ;uh?                                (?)
21    DEBUG_SEND      ;uh?                                (?)
22    RSA                                                 (Port 1000B000h)
23    CTRCARD0        and/or SPI_CARD ?                   (Port 10004000h)
24    CTRCARD1                                           (Port 10005000h?)
25    CGC     Gamecard power off  (CFG9_CARD_PWROFF_DELAY) (Port 10000010h)
26    CGC_DET Gamecard insert     (CFG9_CARD_INSERT_DELAY) (Port 10000010h)
27    NTRCARD (used by ARM9, although it's ARM11/ARM9)    (Port 10164000h)
28    XDMA Event 0..4 (five events sharing one IRQ)       (Port 1000C000h)
29    XDMA Faulting (eg. CCR=0, or event>11)              (Port 1000C000h)
30-31 Unused (always 0)
```

3DS ARM9 Timers

Timers
10003000h ARM9
Same as GBA/NDS/DSi, except, reportedly faster & with stunning accuracy:

```
"timers run at a frequency of 67,027,964.0 +/- 2^(-32) Hz"
```

which would be an estimated error of about +/- 1 cycle per century, or +/- 1 second per a few billions of years.
GBA Timers

3DS ARM11 Interrupts

Private Software Interrupts (can be different for each CPU)

```
IRQ    Listener      Description
00h                  MPCore software IRQ, not configured
01h                  MPCore software IRQ, used by BOOT11 to kickstart Core1
02h-03h              MPCore software IRQ, seem to be unused
04h    Kernel        MPCore software IRQ, used to manage performance counter
05h    Kernel        MPCore software IRQ, does apparently nothing
06h    Kernel        MPCore software IRQ, extensively used by
                       KernelSetState (and contains most of the actual
                       code of the latter)
07h    Kernel        MPCore software IRQ, see KCacheMaintenanceInterruptEvent
08h    Kernel        MPCore software IRQ, used for scheduling
09h    Kernel        MPCore software IRQ, used when handling
                       exceptions that require termination of a thread or
                       a process, and in some cases by
                       svcSetDebugThreadContext, to store VFP registers
                       in the thread's register storage.
0Ah    Kernel        TLB operations IRQ, see KTLBOperationsInterruptEvent
0Bh-0Eh              MPCore software IRQ, not configured
0Fh    dmnt/debugger MPCore software IRQ, used to abstract FIQ (debug),
                       this interrupt is never sent to New3DS core2/core3
```

Private Timer Interrupts (has separate timers for each CPU)

```
10h-1Ch              Hmmmm, these do NOT EXIST (?) in interrupt controller?
1Dh    Kernel        MPCore Timer0 (Port 17E00600h)
1Eh    Kernel        MPCore Timer1 (Port 17E00620h) (3DS used on "core 1")
1Fh                  MPCore Legacy "nIRQ" pin (is that used in 3DS?)
```

Hardware Interrupts (can be used for all CPUs that are selected as Target)

```
20h-23h -           Unused?
24h     spi?        SPI_BUS2 (unused)              (Port 10143000h)
```

```
25h-27h -           Unused?
28h     gsp, TwlBg  PSC0 (GPU_MEMFILL 0)                (Port 10400010h)
29h     gsp, TwlBg  PSC1 (GPU_MEMFILL 1)                (Port 10400020h)
2Ah     gsp, TwlBg  PDC0 (GPU H/V-IRQ for top screen)    (Port 10400400h)
2Bh     gsp, TwlBg  PDC1 (GPU H/V-IRQ for bottom screen) (Port 10400500h)
2Ch     gsp, TwlBg  PPF  (GPU_MEMCOPY)                   (Port 10400C00h)
2Dh     gsp, TwlBg  P3D  (GPUREG_IRQ_CMP/REQ)            (Port 10401040h)
2Eh-2Fh -           Unused?
30h-38h Kernel      Old CDMA Event 0..8 (nine events with separate IRQs)
39h     Kernel      Old CDMA Faulting (eg. CCR=0, or event>15)
3Ah     Kernel      New CDMA Event 0..31 (32 events sharing one IRQ) ;\New3DS
3Bh     Kernel      New CDMA Faulting (eg. CCR=0)                    ;/
3Ch-3Fh -           Unused?
40h     nwm         WIFI SDIO Controller            (Port 10122000h)
41h     nwm         WIFI SDIO IRQ Pin
42h     nwm_dev?    Debug WIFI SDIO Controller      (Port 10100000h?)
43h     -           Unused? Or maybe Debug WIFI SDIO IRQ Pin for above?
44h     -           NTRCARD                         (Port 10164000h)
45h     mvd         L2B_0 (First RGB-to-RGBA Converter)  (10130000h) ;\New3DS
46h     mvd         L2B_1 (Second RGB-to-RGBA Converter) (10131000h) ;/
47h     -           Unused?
48h     camera      Camera Bus 0 (DSi cameras)      (Port 10120000h)
49h     camera      Camera Bus 1 (left-eye)         (Port 10121000h)
4Ah     dsp         ...probably Teak DSP... ?      (maybe 10203000h)
4Bh     camera      Y2R_0 (First YUV-to-RGBA Converter)  (10102000h)
4Ch     TwlBg       LGYFB_0 Legacy GBA/NDS Video    (Port 10110000h)
4Dh     TwlBg       LGYFB_1 Legacy GBA/NDS Video    (Port 10111000h)
4Eh     mvd         Y2R_1 (Second YUV-to-RGBA Converter) (10132000h) ;\New3DS
4Fh     mvd         MVD Registers                   (Port 10207000h) ;/
50h     pxi, TwlBg  PXI_SYNC.bit29 from ARM9 (commonly used)
51h     pxi, TwlBg  PXI_SYNC.bit30 from ARM9 (rarely used)
52h     pxi, TwlBg  PXI Send Fifo Empty
53h     pxi, TwlBg  PXI Receive Fifo Not Empty
54h     i2c, TwlBg  I2C_BUS0 (DSi devices)          (Port 10161000h)
55h     i2c, TwlBg  I2C_BUS1 (3DS devices)          (Port 10144000h)
56h     spi, TwlBg  SPI_BUS0 (Pwrman,WifiFlash,Tsc) (Port 10160000h)
57h     spi, TwlBg  SPI_BUS1 (Tsc)                  (Port 10142000h)
58h     Kernel      PDN (see CFG11_MPCORE_CLKCNT) (maybe other sources, too)
59h     TwlBg       ?
5Ah     mic         Microphone maybe?              (maybe 10162000h)
5Bh     -           HID PAD Controller Buttons      (Port 10146000h)
5Ch     i2c, TwlBg  I2C_BUS2 (3DS extra gimmicks)   (Port 10148000h)
5Dh-5Eh -           Unused?
5Fh     -           NDS-Wifi Registers (aka MP)     (Port 10170000h)
60h     gpio, TwlBg GPIO_DATA0.bit2?   Shell opened
61h     -           Unused?
62h     gpio, TwlBg GPIO_DATA0.bit2?   Shell closed
63h     gpio, TwlBg GPIO_DATA0.bit1    Touchscreen Pen Down (if enabled)
64h     gpio, TwlBg GPIO_DATA1.bit0    Headphone jack plugged in/out
65h     -           Unused?
66h     gpio, TwlBg GPIO_DATA1.bit1    ?
67h     -           Unused?
68h     gpio, TwlBg GPIO_DATA3.bit0(?) C-stick Interrupt (New3DS)
69h     gpio, TwlBg GPIO_DATA3.bit1    IrDA Interrupt
6Ah     gpio, TwlBg GPIO_DATA3.bit2(?) Gyro Interrupt
6Bh     gpio, TwlBg GPIO_DATA3.bit3    ?
6Ch     gpio, TwlBg GPIO_DATA3.bit4    ?
6Dh     gpio, TwlBg GPIO_DATA3.bit5    ?
6Eh     gpio, TwlBg GPIO_DATA3.bit6    ?
6Fh     gpio, TwlBg GPIO_DATA3.bit7    ?
70h     gpio, TwlBg GPIO_DATA3.bit8(?) ?
71h     gpio, TwlBg GPIO_DATA3.bit9(?) MCU Interrupt (HOME/POWER button etc)
72h     gpio, TwlBg GPIO_DATA3.bit10   NFC Interrupt (New3DS)
73h     TwlBg       GPIO_DATA3.bit11(?)??
74h     ?           CGC Gamecard power off     (CFG9_CARD_PWROFF_DELAY)
```

```
  75h     ?           CGC Gamecard insert switch (CFG9_CARD_INSERT_DELAY)
  76h     -           L2C Level 2 Cache Controller    (Port 17E10000h)   New3DS
  77h     -           Unused?
  78h     Kernel      CPU0 cp15 Performance monitor count (any) overflow
  79h     Kernel      CPU1 cp15 Performance monitor count (any) overflow
  7Ah     Kernel      CPU2 cp15 Performance monitor count (any) overflow New3DS
  7Bh     Kernel      CPU3 cp15 Performance monitor count (any) overflow New3DS
  7Ch-7Fh -           Unused?
  80h-3FEh            Don't exist (3DS/New3DS has only 80h IRQ sources)
  3FFh                None, no interrupt (or spurious interrupt)
```

The spurious interrupt (3FFh) might appear in an interrupt handler if the handler got triggered, but something (eg. another CPU) has cleared the IRQ flag before the handler got a chance to process it.

ARM11 MPCore Private Memory Region Register Summary

MPCore private memory region

The MPCore Private Memory Region contains additional ARM control registers (additionally to the CP15 coprocessor registers). The 3DS does have this region mapped at address 17E00000h and up.

```
  17E00000h 100h  MPCore SCU (Snoop Control Unit)
  17E00100h 100h  MPCore CPU interrupt interface for LOCAL CPU
  17E00200h 100h  MPCore CPU0 interrupt interface (aliased for debug purposes)
  17E00300h 100h  MPCore CPU1 interrupt interface (aliased for debug purposes)
  17E00400h 100h  MPCore CPU2 interrupt interface (aliased for debug purposes)
  17E00500h 100h  MPCore CPU3 interrupt interface (aliased for debug purposes)
  17E00600h 100h  MPCore CPU timer and watchdog for LOCAL CPU
  17E00700h 100h  MPCore CPU0 timer and watchdog
  17E00800h 100h  MPCore CPU1 timer and watchdog
  17E00900h 100h  MPCore CPU2 timer and watchdog  ;\if enabled in
  17E00A00h 100h  MPCore CPU3 timer and watchdog  ;/10141312h/13h
  17E00B00h 500h  MPCore Reserved (access causes a DECERR abort exception)
  17E01000h 1000h MPCore Global Interrupt distributor
```

Snoop Control Unit (SCU)

```
  Offset       Type Reset      Name
  17E00000h 4   R/W 00001FFEh  SCU Control Register
  17E00004h 4   R   (var)      SCU Configuration Register
  17E00008h 4   R/W -          SCU CPU Status
  17E0000Ch 4   W   -          SCU Invalidate All
  17E00010h 4   R/W 00000000h  SCU Performance Monitor Control Register
  17E00014h 4   R/W 00000000h  SCU Monitor Counter Events 0
  17E00018h 4   R/W 00000000h  SCU Monitor Counter Events 1
  17E0001Ch 4   R/W 00000000h  SCU Monitor Counter 0
  17E00020h 4   R/W 00000000h  SCU Monitor Counter 1
  17E00024h 4   R/W 00000000h  SCU Monitor Counter 2
  17E00028h 4   R/W 00000000h  SCU Monitor Counter 3
  17E0002Ch 4   R/W 00000000h  SCU Monitor Counter 4
  17E00030h 4   R/W 00000000h  SCU Monitor Counter 5
  17E00034h 4   R/W 00000000h  SCU Monitor Counter 6
  17E00038h 4   R/W 00000000h  SCU Monitor Counter 7
  17E0003Ch C4h -   -          SCU Reserved (0)
```

All SCU registers are byte accessible.

MP11 CPU Interrupt Interface Registers (for Interrupt Handling)

```
  Address       Type Reset value Function
  17E00100h 4   R/W  00000000h CPU IRQ Control Register
  17E00104h 4   R/W  000000F0h CPU IRQ Priority Mask Register
  17E00108h 4   R/W  00000003h CPU IRQ Binary Point Register
  17E0010Ch 4   R    000003FFh CPU IRQ Interrupt Acknowledge?? Register
  17E00110h 4   W    -         CPU IRQ End of Interrupt Register
  17E00114h 4   R    000000F0h CPU IRQ Running Priority Register
```

```
  17E00118h 4   R    000003FFh CPU IRQ Highest Pending Interrupt Register
  17E0011Ch E4h -    -         CPU IRQ Reserved
```
Above registers are for the LOCAL CPU, below are aliases for EACH CPU core.
```
  17E00200h 100h   MPCore CPU0 interrupt interface (aliased for debug purposes)
  17E00300h 100h   MPCore CPU1 interrupt interface (aliased for debug purposes)
  17E00400h 100h   MPCore CPU2 interrupt interface (aliased for debug purposes)
  17E00500h 100h   MPCore CPU3 interrupt interface (aliased for debug purposes)
```
All registers of the MP11 CPU interrupt interfaces must be accessed by 32bit transactions only.

Timer and Watchdog registers
```
  Offset       Type Reset     Name
  17E00600h 4   R/W 00000000h MPCore Timer0 Reload Value
  17E00604h 4   R/W 00000000h MPCore Timer0 Counter Value (decrementing)
  17E00608h 4   R/W 00000000h MPCore Timer0 Control Register
  17E0060Ch 4   R/W 00000000h MPCore Timer0 Interrupt Status
  17E00610h 10h -             Reserved
  17E00620h 4   R/W 00000000h MPCore Timer1/Watchdog Reload Value
  17E00624h 4   R/W 00000000h MPCore Timer1/Watchdog Counter Value (decrem.)
  17E00628h 4   R/W 00000000h MPCore Timer1/Watchdog Control Register
  17E0062Ch 4   R/W 00000000h MPCore Timer1/Watchdog Interrupt Status
  17E00630h 4   R/W 00000000h MPCore Timer1/Watchdog Reset Sent Register
  17E00634h 4   W   -         MPCore Timer1/Watchdog Disable Register
  17E00638h C8h -             Reserved
```
Above registers are for the LOCAL CPU, below are aliases for EACH CPU core.
```
  17E00700h 100h              MPCore CPU0 timer and watchdog
  17E00800h 100h              MPCore CPU1 timer and watchdog
  17E00900h 100h              MPCore CPU2 timer and watchdog  ;\if enabled in
  17E00A00h 100h              MPCore CPU3 timer and watchdog  ;/10141312h/13h
```
All timer and watchdog registers are word accessible only.
Note: There's also another ARM11 timer/cycle counter in CP15 registers.

Distributed Interrupt controller registers (for Interrupt Configuration)
```
  Address   Size  Type Reset     Function
  17E01000h 4     R/W  00000000h Interrupt Distributor Control Register
  17E01004h 4     R    -         Interrupt Controller Type Register
  17E01008h F8h   -    ....      Reserved
  17E01100h 20h   R/W  0000FFFFh Interrupt Enable set Registers ID0-ID31
  17E01104h ()          00000000h Interrupt Enable set Registers ID32 and up
  17E01120h 60h   -    ....      Reserved
  17E01180h 20h   R/W  0000FFFFh Interrupt Enable clear Registers ID0-ID31
  17E01184h ()          00000000h Interrupt Enable clear Registers ID32 and up
  17E011A0h 60h   -    ....      Reserved
  17E01200h 20h   R/W  00000000h Interrupt Pending set Registers
  17E01220h 60h   -    ....      Reserved
  17E01280h 20h   R/W  00000000h Interrupt Pending clear Registers
  17E012A0h 60h   -    ....      Reserved
  17E01300h 20h   R    00000000h Interrupt Active Bit Registers
  17E01320h E0h   -    ....      Reserved
  17E01400h 100h  R/W  00000000h Interrupt Priority Registers
  17E01500h 300h  -    ....      Reserved
  17E01800h 100h  R/W  00000000h Interrupt CPU targets Registers (a.)
  17E01900h 300h  -    ....      Reserved
  17E01C00h 40h   R/W  AAAAAAAAh Interrupt Configuration Registers ID0-ID15
  17E01C04h ()          28000000h Interrupt Configuration Registers ID29-ID31
  17E01C08h ()          00000000h Interrupt Configuration Registers ID32 and up
  17E01C40h C0h   -    ....      Reserved
  17E01D00h 20h   R    00000000h Interrupt Line Level Registers ID0-ID31
  17E01D04h ()          -         Interrupt Line Level Registers ID32 and up
  17E01D20h E0h   -    ....      Oddly: mirrors of above 20h bytes
  17E01E00h 100h  -    ....      Reserved
  17E01F00h 4     W    -         Software Interrupt Register
  17E01F0xh DCh?  -    ....      Reserved
  17E01FE0h 4     R    90h       Peripheral Identification Register 0
```

```
  17E01FE4h 4     R    13h       Peripheral Identification Register 1
  17E01FE8h 4     R    04h       Peripheral Identification Register 2
  17E01FECh 4     R    00h       Peripheral Identification Register 3
  17E01FF0h 4     R    0Dh       PrimeCell Identification Register 0
  17E01FF4h 4     R    F0h       PrimeCell Identification Register 1
  17E01FF8h 4     R    05h       PrimeCell Identification Register 2
  17E01FFCh 4     R    B1h       PrimeCell Identification Register 3
    a. Except for address 0x81C. See Interrupt CPU Targets Registers.
```
All Interrupt Distributor Registers are byte accessible.

Official specs: DDI0360F_arm11_mpcore_r2p0_trm.pdf

ARM11 MPCore - Snoop Control Unit (SCU)

Snoop Control Unit (SCU)
Below registers exists only once (not per CPU). However, the performance counters are somewhat supposed to be used as two counters per CPU.

17E00000h - SCU Control Register (R/W)
The SCU Control Register enables the SCU and controls its behavior. It must be accessed using a read-modify-write sequence.
```
  0     SCU Enable      (0=Disable, 1=Enable)
        (enable: maintain coherency between MP11 CPUs Level 1 data side caches)
        (in single CPU configuration, this bit has no effect and is always 0)
  1-4   Allow CPU0..3 to access SCU at 17E00000h..17E000FFh (0=No, 1=Yes)
  5-8   Allow CPU0..3 to access TMR at 17E00200h..17E005FFh (0=No, 1=Yes)
  9-12  Allow CPU0..3 to access IRQ at 17E00700h..17E00AFFh (0=No, 1=Yes)
  13    Report RAM Parity errors via parity error signals (0=Disable, 1=Enable)
  14-31 Reserved SBZ
```
Bit1-4: There is a mechanism that prevents all bits being cleared at the same time.
Bit13: Before enabling SCU parity checking, all SCU tag must be invalidated.
This register doesn't seem to allow to disable access to the Global Interrupt Distributor at 17E01000h-17E01FFFh (however, the MMU can disable the whole 4Kbyte page).

17E00004h - SCU Configuration Register (R)
```
  0-1   Number of ARM11 CPU cores   (0..3 = 1,2,3,4 CPU's)
  2-3   Reserved SBZ
  4-7   CPU0..3 Symmetric/Asymmetric Multi-processing (0=SMP/coherent, 1=AMP)
  8-9   CPU0 Tag RAM cache size (0=16KB, 1=32KB, 2=64KB, 3=Reserved)
  10-11 CPU1 Tag RAM cache size (0=16KB, 1=32KB, 2=64KB, 3=Reserved)
  12-13 CPU2 Tag RAM cache size (0=16KB, 1=32KB, 2=64KB, 3=Reserved)
  14-15 CPU3 Tag RAM cache size (0=16KB, 1=32KB, 2=64KB, 3=Reserved)
  16-31 Reserved SBZ
```
Cache size 16KB/32KB/64KB implies 64/128/256 indexes per tag RAM accordingly.
On New3DS this is 00005013h (even in Old3DS mode). But changes to 00005003h when using fastboot...?

17E00008h - SCU CPU Status Register (R/W)
```
  0-1   CPU0 status (0=Normal, 1=Reserved, 2=Dormant, 3=Powered-off)
  2-3   CPU1 status (0=Normal, 1=Reserved, 2=Dormant, 3=Powered-off)
  4-5   CPU2 status (0=Normal, 1=Reserved, 2=Dormant, 3=Powered-off)
  6-7   CPU3 status (0=Normal, 1=Reserved, 2=Dormant, 3=Powered-off)
  8-31  Reserved SBZ
```
Dormant mode and powered-off mode are controlled by an external power controller.
SCU CPU Status Register bits indicate to the external power controller which power domains can be powered down.
Before entering any other power mode than Normal, the MP11 CPU must set its status field to signal to the SCU which mode it is about to enter (so that the SCU can determine if it still can send coherency requests to the

CPU). The MP11 CPU then executes a WFI entry instruction. When in WFI state, the PWRCTLOn bus is enabled and signals to the power controller what it must do with power domains.
The SCU CPU Status Register bits are used in conjunction with internal WFI entry signals to generate PWRCTLOn output pins.
The SCU CPU Status Register bits can also be read by a CPU exiting low-power mode to determine its state before executing its reset setup.
MP11 CPUs status fields take PWRCTLIn values at reset, except for nonpresent CPUs.
For nonpresent CPUs writing to this field has no effect.

17E0000Ch - SCU Invalidate All Register (W)

```
0-3   Invalidate CPU0 ways (bit0-3   = Way 0,1,2,3)  (0=No, 1=Invalidate)
4-9   Invalidate CPU1 ways (bit4-9   = Way 0,1,2,3)  (0=No, 1=Invalidate)
8-11  Invalidate CPU2 ways (bit8-11  = Way 0,1,2,3)  (0=No, 1=Invalidate)
12-15 Invalidate CPU3 ways (bit12-15 = Way 0,1,2,3)  (0=No, 1=Invalidate)
16-31 Reserved SBZ
```

Allows to invalidate the tag RAMs on a per CPU and per way basis. This operation is atomic, that is, a write transfer to this address only terminates when all the lines have been invalidated. This register reads as 0.
Uh, what is a "way"?

17E00010h - Performance Monitor Control Register (R/W)

```
0      Enable bit for all counters     (0=Disable, 1=Enable)
1      Reset all count registers       (0=No, 1=Reset)
2-7    Reserved SBZ
8-15   Counter MN0..7 Interrupt Enable (0=Disable, 1=Enable)
16-23  Counter MN0..7 Interrupt Flag (0=No, 1=Overflow/IRQ) ;write 1 to clear
24-31  Reserved SBZ/RAZ
```

17E00014h - SCU Monitor Counter Events 0, bit0-31 (R/W)
17E00018h - SCU Monitor Counter Events 1, bit32-63 (R/W)

```
0-7     EvCount0  Identifies the event for counter MN0
8-15    EvCount1  Identifies the event for counter MN1
16-23   EvCount2  Identifies the event for counter MN2 (if any)
24-31   EvCount3  Identifies the event for counter MN3 (if any)
48-55   EvCount6  Identifies the event for counter MN6 (if any)
56-63   EvCount7  Identifies the event for counter MN7 (if any)
32-39   EvCount4  Identifies the event for counter MN4 (if any)
40-47   EvCount5  Identifies the event for counter MN5 (if any)
```

Event source number definitions:

```
00h     Counter disabled
01h     CPU0 Miss  ;\
02h     CPU1 Miss  ; CPUn requested a coherent linefill that misses in all
03h     CPU2 Miss  ; other CPUs. The request is sent to external memory
04h     CPU3 Miss  ;/
05h     CPU0 Hit   ;\
06h     CPU1 Hit   ; CPUn requested a coherent linefill that hits in another
07h     CPU2 Hit   ; CPU. The linefill is fetched from the relevant CPU cache
08h     CPU3 Hit   ;/
09h     CPU0 Error ;\
0Ah     CPU1 Error ; CPUn was expected to have a coherent
0Bh     CPU2 Error ; line in its cache but answers nonpresent.
0Ch     CPU3 Error ;/
0Dh     Line migration  ;-A line is directly transferred from one
                        ;  CPU to another on a linefill request instead of
                        ;  switching to SHARED.
0Eh     Master0 Read port busy
0Fh     Master1 Read port busy
10h     Master0 Write port busy
11h     Master1 Write port busy
12h     A Read transfer is sent to the external memory
13h     A Write transfer is sent to the external memory
14h-1Eh N/A
```

```
  1Fh     CycleCount   ;-The counter increments on each CPU clock cycle
  20h-FFh N/A
```

17E0001Ch - SCU Monitor Counter MN0 (R/W) ;exists always
17E00020h - SCU Monitor Counter MN1 (R/W) ;exists always
17E00024h - SCU Monitor Counter MN2 (R/W) ;exists only for two or more CPUs
17E00028h - SCU Monitor Counter MN3 (R/W) ;exists only for two or more CPUs
17E0002Ch - SCU Monitor Counter MN4 (R/W) ;exists only for three or more CPUs
17E00030h - SCU Monitor Counter MN5 (R/W) ;exists only for three or more CPUs
17E00034h - SCU Monitor Counter MN6 (R/W) ;exists only for four CPUs
17E00038h - SCU Monitor Counter MN7 (R/W) ;exists only for four CPUs
```
  0-31  Counter (incrementing upon selected event)
```
New3DS has eight counters (even in Old3DS mode).

ARM11 MPCore - Timer and Watchdog

Timer and Watchdog
Below registers are for the LOCAL CPU core (the other CPU cores have their own registers, mapped at the same address).

17E00600h - MPCore Timer0 Reload Value (R/W)
17E00620h - MPCore Timer1/Watchdog Reload Value (R/W)
```
  0-31  Reload Value
```
The Reload value is copied to the Counter in two situations:
1) When the Counter decrements to zero (with Auto-reload enabled).
2) When writing to the Reload register (in watchdog mode this is the only way to refresh the watchdog, and to prevent it from resetting the CPU).

17E00604h - MPCore Timer0 Counter Value (R/W)
17E00624h - MPCore Timer1/Watchdog Counter Value (R/W) (R in watchdog mode)
```
  0-31  Counter Value (decrementing)
```
If the MP11 CPU belonging to the timer is in debug state, the counter does not decrement until the MP11 CPU returns to non debug state.

17E00608h - MPCore Timer0 Control Register (R/W)
17E00628h - MPCore Timer1/Watchdog Control Register (R/W)
```
  0     Timer Enable     (0=Stop, 1=Enable/Decrement)
  1     Auto-reload      (0=One-shot, stop at zero, 1=Auto-reload at zero)
  2     Interrupt Enable (0=Disable, 1=Trigger Interrupt ID 29/30 at zero)
  3     Timer0: Reserved      (0=Timer, fixed, always 0) (R)
  3     Timer1: Watchdog mode (0=Timer, 1=Watchdog)      (R or R/W)
          Note: Bit3 can be cleared via 17E00634h only
  4-7   Reserved (0)
  8-15  Prescaler (0..255 = CPU_CLK/2 divided by 1..256)
  16-31 Reserved (0)
```

17E0060Ch - MPCore Timer0 Interrupt Status (R/ack)
17E0062Ch - MPCore Timer1/Watchdog Interrupt Status (R/ack)
```
  0     Event flag (counter reached zero)  (0=No, 1=Event)  ;write 1 to clear
  1-31  Reserved
```
If the timer interrupt is enabled, Interrupt ID 29/30 is set as Pending in the Interrupt Distributor after the event flag is set.

17E00630h - MPCore Timer1/Watchdog Reset Sent Register (R/ack)
```
  0     Reset flag  (0=Normal, 1=Reset caused by Watchdot)  ;write 1 to clear
```

```
  1-31  Reserved
```

17E00634h - W - MPCore Timer1/Watchdog Disable Register

```
  0-31  Key (write 12345678h, then write 87654321h to disable watchdog)
```

Switches the watchdog back to Timer mode (ie. clears bit3 in Control Register).

Calculating timer intervals

```
 The timer interval is calculated using the following equation:
   ((PRESCALER_value+1)*(Reload_value+1)*x2 / CPU_CLK_frequency
 This equation can be used to calculate the period between two events out of
 the timers and the watchdog time-out time.
```

Uh, doesn't that mean that reload occurs on UNDERFLOW (not on ZERO)...?

ARM11 MPCore - Interrupt Configuration

Distributed Interrupt Controller (for Interrupt Configuration)

These registers are mostly used for interrupt configuration (and for internally maintaining an list of pending/active interrupts).

```
  -- Most of these registers exists only once (not per CPU) --
```

Except, the Priority for Software Interrupts 00h-0Fh can be configured per LOCAL CPU. And, the Pending and Active flags do <internally> exist for each CPU (for Software Interrupts 00h-0Fh, they do even exists for "From each CPU to each CPU"). However, reading the Pending/Active/Priority flags doesn't fully represent all of the internal per-CPU states.

17E01000h - Interrupt Distributor Control Register (R/W)

```
  0     Global Interrupt Controller Enable    (0=Disable, 1=Enable)
  1-31  Reserved
```

If bit0=0, no interrupts at all are sent to the CPU interrupt interfaces.

17E01004h - Interrupt Controller Type Register (R)

```
  0-4   Number of Interrupt IDs   (0-7 = 20h,40h,60h,...,100h) (8-31=Reserved)
  5-7   Number of ARM11 CPU cores (0-3 = 1,2,3,4 CPU's)        (4-7=Reserved)
  8-31  Reserved
```

Note: Interrupt ID 00h-1Fh are internal IRQs, ID 20h-FFh are external IRQs.
On New3DS this 00000063h, 4 CPUs and 80h IRQs (even in Old3DS mode).

17E01100h..17E0111Fh - Interrupt Enable Set Registers (256 x 1bit) (R/W)
17E01180h..17E0119Fh - Interrupt Enable Clear Registers (256 x 1bit) (R/W)

```
  0-15    Interrupt 00h-0Fh (Read: Always 1=Enabled)      (Write=No effect)
  16-255  Interrupt 10h-FFh (Read: 0=Disabled, 1=Enabled) (Write: 1=Set/Clear)
```

Enable means that pending IRQs will be transmitted to the targeted CPUs.
The enable bit, when set to 0, does not prevent an edge-triggered interrupt from becoming Pending.
The enable bit, when set to 0, does prevent a level sensitive interrupt from becoming Pending only if asserted by the hardware pin, INT.
Note: If an interrupt is Pending or Active when its enable bit is set to 0, it remains in its current state.

17E01200h..17E0121Fh - Interrupt Pending set Registers (256 x 1bit) (R/W)
17E01280h..17E0129Fh - Interrupt Pending clear Registers (256 x 1bit) (R/W)

```
  0-15   Interrupt 00h-0Fh pending on local CPU (0=No, 1=Pending) (W=??)
  16-31  Interrupt 10h-1Fh pending on local CPU (0=No, 1=Pending) (W=?)
  32-255 Interrupt 20h-FFh pending on CPU(s)??  (0=No, 1=Pending) (W=Set/Clr)
 Bit0-31:   Reading returns 1 if pending on local CPU core
 Bit32-255: Reading returns 1 if pending on one or more ANY CPU cores
             (if it's pending on the LOCAL CPU can be seen only once when
             reading the Acknowledge register?)
             (the Enable and Target bits do also somewhat imply whether
```

```
            the interrupt could have become pending on local CPU)
 Bit0-15:   Write/Set is ignored (use 17E01F00h Software Interrupt instead)
 Bit16-31:  Write/Set is...?
 Bit32-255: Write/Set allows to force the state=1 for ALL TARGET CPU cores
 Bit0-15:   Write/Clear is...?
 Bit16-31:  Write/Clear is...?
 Bit32-255: Write/Clear allows to force the state=0 for ALL CPU cores
```

17E01300h..17E0131Fh - Active Bit Registers (256 x 1bit) (R)

```
  0-31   Interrupt 00h-1Fh is active on local CPU       (0=No, 1=Active)
  32-255 Interrupt 20h-FFh is active on one or more CPUs (0=No, 1=Active)
```

Active means that the interrupt is being processed on at least one MP11 CPU.

17E01400h..17E014FFh - Interrupt Priority Registers (256 x 8bit) (R/W)

The priority determines which interrupt will show up next in the interrupt interface registers. And, the interface registers can be used to manually disable lower priority interrupts during 'critical sections'. And, while processing an interrupt, the hardware will automatically disable interrupts with same or lower priority (optionally, the "Binary Point" feature allows to ignore the LSBs of the priorioty, eg. ignoring the lowest bit with allow to also disable interrupts with slightly higher priority).

```
  0-127     Interrupt 00h-0Fh Priority for local CPU (bit0-3=SBZ, bit4-7=Prio)
  128-231   Interrupt 10h-1Ch Priority for what?     (bit0-3=SBZ, bit4-7=Prio)
  232-255   Interrupt 1Dh-1Fh Priority for local CPU (bit0-3=SBZ, bit4-7=Prio)
  256-2047  Interrupt 20h-FFh Priority for all CPUs  (bit0-3=SBZ, bit4-7=Prio)
```

Priority values are 00h=Highest... E0h=Lowest, or F0h=None (same as interrupt disabled).
When multiple Pending interrupts have the same priority, the selected interrupt is the one with lowest ID. If there are multiple Pending software interrupts with the same ID, the lowest MP11 CPU source is selected.

17E01800h..17E018FFh - Interrupt CPU Targets Registers (256 x 8bit) (R/W)

These registers store the list of MP11 CPUs for which an interrupt can be Pending. Interrupt target registers are ignored in cases of software triggered interrupts.

```
  0-231     Interrupt 00h-1Ch Target (fixed 00h) (see 17E01F00h instead)  (-)
  232-255   Interrupt 1Dh-1Fh Target (fixed 01h/02h/04h/08h for CPU0..3)  (R)
  256-2047  Interrupt 20h-FFh Target (bit0-3=CPU0..3, bit4-7=SBZ)       (R/W)
```

Modifying a CPU target list has no influence on a Pending or Active interrupt, but takes effect on a subsequent assertion of the interrupt.

17E01C00h..17E01C3Fh - Interrupt Configuration Registers (256 x 2bit) (R/W)

Interrupt Configuration Registers define the assertion condition and the software model of each interrupt.

```
  0-31     Interrupt 00h-0Fh Condition (can be 2,3) (always rising-edge)
  32-57    Interrupt 10h-1Ch Condition (always 0)
  58-61    Interrupt 1Dh-1Eh Condition (always 2) (always N-N, rising-edge)
  62-63    Interrupt 1Fh     Condition (always 0) (ignored, always low active)
  64-511   Interrupt 20h-FFh Condition (can be 0,1,2,3)
```

Interrupt line encodings for bits 1 and 0:

```
  00h = N-N software model, level high active
  01h = 1-N software model, level high active
  02h = N-N software model, rising edge sensitive
  03h = 1-N software model, rising edge sensitive
```

The N-N and 1-N models are relevant only if more than one CPU is configured as Target:

```
  1-N model: An interrupt that is taken on any CPU clears the Pending status
    on all CPUs.
  N-N model: All CPUs receive the interrupt independently. The Pending status
    is cleared only for the CPU that takes it, not for the other CPUs. The N-N
    model has been deprecated in the latest interrupt controller architecture.
```

Unknown how 1-N and N-N differ for Interrupt 00h-0Fh (it's said that each CPU has it's own pending flags for 00h-0Fh, so the model should be always "1-1").

```
  Obscure Notes:
  - With the 1-N software model, the nIRQ input is asserted on all CPUs
    configured in the CPU Targets Register. Uh, but Interrupt 1Fh is fixed?
  - If more than one of these CPUs reads the Interrupt Acknowledge Register
```

```
at the same time, they can all acknowledge the same interrupt. The
interrupt service routine must ensure that only one of them tries to
process the interrupt, with the others returning after writing the ID to
the End of Interrupt Register. Uh, does that apply to nIRQ only? Or to
all IRQs?
```

17E01D00h..17E01D1Fh - Interrupt Line Level Registers (256 x 1bit) (R)
```
0-31    Dummy line level bits for Internal IRQ 00h..1Fh (always 0)
32-255  Interrupt line level for External IRQ 20h..FFh  (0=Low, 1=High)
```

17E01F00h - Software Interrupt Register (W)
```
0-9     Interrupt ID (0..FFh? or 0..0Fh/1Fh?) (100h..1FFh=Reserved/ignored)
10-15   SBZ
16-19   Send the interrupt to CPU0..3 (0=No, 1=Yes) ;used only if Bit24-25=0
20-23   SBZ
24-25   Target list mode (0=Bit16-19, 1=Other CPUs, 2=Local CPU, 3=Reserved)
26-31   SBZ
```
This can be used to trigger an interrupt (identified with its ID) to a list of MP11 CPUs.

ARM11 MPCore - Interrupt Handling

CPU Interrupt Interface Registers (for Interrupt Handling)
Below registers are for the LOCAL CPU core (the other CPU cores have their own registers, mapped at the same address).

17E00100h - CPU Interface Control Register (R/W)
```
0     Interrupt Enable for local CPU   (0=Disable, 1=Enable)
1-31  Reserved
```
When disabled: External nIRQ input is still working (and, in that case, nIRQ should be handled directly; without using the disabled ACK/EOI registers).

17E00104h - Priority Mask Register (R/W)
The priority mask is used to prevent interrupts from being sent to the MP11 CPU. The CPU Interface asserts an interrupt request to an MP11 CPU if the priority of the highest Pending interrupt sent by the Interrupt Distributor is strictly higher than the mask set in the Priority Mask Register.
```
0-3  Unused SBZ
4-7  Priority Mask value
          NOTE: This is a 4bit COMPARE value (not an AND-mask)
             0xF Interrupts with priority 0x0-0xE are not masked.
             0x0 All interrupts are masked.
8-31 Unused SBZ
```

17E00108h - Binary Point Register (R/W)
The Binary Point Register is used to determime whether a new interrupt pre-empts a currently Active one, using only part, or none of the priority level.
```
0-2   Binary Point (see below)
3-31  Reserved
```
Binary point bit values assignment:
```
03h = All bits4,5,6,7 of priority are compared for pre-emption
04h = Only bit5,6,7   of priority are compared for pre-emption
05h = Only bit6,7     of priority are compared for pre-emption
06h = Only bit7       of priority is compared for pre-emption
07h = No bits compared, no pre-emption is performed (no nested IRQs)
00h,01h,02h = Same as 03h
```
This is related to the "Running Priority Register" (see there for details).

17E0010Ch - Interrupt Acknowledge (ACK) Register (R)

```
  0-9     Interrupt ID          (00h..FFh, or 3FFh=None)
  10-12   Source for ID=00h-0Fh (0..3=CPU0..3, or always 0 for ID=10h-3FFh)
  13-31   Unused (0)
```
Reading returns the ID of the next pending interrupt with highest priority, and automatically switches the interrupt from pending state to active state (indicating that it is being processed). The CPU should then process the interrupt, and, thereafer, use End of Interrupt register (to indicate that processing is done).

17E00110h - End of Interrupt (EOI) Register (W)
```
  0-31    Same format as Interrupt Acknowledge Register (see there)
```
Writing clears the Active flag for the corresponding interrupt, indicating that interrupt processing is done. The written value should be same as the (memorized) value from the Interrupt Acknowledge Register.

17E00114h - Running Priority Register (R)
Indicates the priority of the currently processed interrupt (the last acknowledged and not yet completed interrupt on the local CPU core).
```
  0-3  Reserved SBZ
  4-7  Priority (0=Highest .. 14=lowest, 15=None/No interrupt being processed)
  8-31 Reserved
```
All interrupts with same or lower priority are automatically disabled until finishing the current interrupt (by writing to End of Interrupt register).
Optionally, the "Binary Point Register" allows to ignore the priority LSB(s) when comparing the new Interrupt's priority with the currently "Running Priority" (for example, ignoring the lower two bits would cause an interrupt with priority=5 to also disable slighly higher priority interrupts with priority=6..7).

17E00118h - Highest Pending Interrupt Register (R)
```
  0-31    Same format as Interrupt Acknowledge Register (see there)
```
Same as Interrupt Acknowledge Register, except that reading doesn't make the corresponding interrupt Active. Uh, and supposedly doesn't clear its Pending state... or does it?

ARM11 MPCore Distributed Interrupt Controller (Blurb)

10.1.1 Distributed Interrupt Controller clock frequency
The Distributed Interrupt Controller logic is clocked at half the frequency of the MPCore CPUs because of power and area considerations. Reducing clock speed reduces dynamic power consumption. The lower clock speed requires less pipelining in the design. This means that the overall impact of the reduced clock speed on the Distributed Interrupt Controller is kept to a minimum.
Note
As a consequence, the minimum pulse width of signals driving external interrupt lines is two CPU clock cycles.

10.2 Terminology - From point of view of an MP11 CPU, an interrupt can be:
```
  Inactive: An Inactive interrupt is one that is nonasserted, or which in a
    multi-processing environment has been completely processed by that MP11
    CPU but can still be either Pending or Active in some of the MP11 CPUs to
    which it is targeted, and so might not have been cleared at the interrupt
    source.
  Pending: A Pending interrupt is one that has been asserted, and for which
    processing has not started on that MP11 CPU.
  Active: An Active interrupt is one that has been started on that MP11 CPU,
    but processing is not complete.
```
An interrupt can be Pending and Active at the same time. This can happen in the case of edge triggered interrupts, when the interrupt is asserted while the MP11 CPU has not finished handling the first occurrence. For level-sensitive interrupts it can only happen if software triggers it. See Interrupt Configuration Registers, 0xC00-0xC3C on page 10-17.
```
  Pre-emption: An Active interrupt can be pre-empted when a new interrupt of
    higher priority interrupts MP11 CPU interrupt processing. For the purpose
```

of this document, an Active interrupt can be running if it is actually being processed, or pre-empted.

The Distributed Interrupt Controller consists of:

Interrupt Distributor:

The Interrupt Distributor handles interrupt detection and interrupt prioritization.

CPU interrupt interfaces:

There is one CPU interrupt interface per MP11 CPU. The MP11 CPU interrupt interfaces handle interrupt acknowledgement, interrupt masking, and interrupt completion acknowledgement.

10.3 Interrupt Distributor

The Interrupt Distributor centralizes all interrupt sources for the ARM11 MPCore processor before dispatching the highest priority ones to each individual MP11 CPU.

All interrupt sources are identified by a unique ID. All interrupt sources have their own configurable priority and list of targeted CPUs, that is, a list of CPUs to which the interrupt is sent when triggered by the Interrupt Distributor.

Note

nFIQ interrupts are not handled by the Distributed Interrupt Controller so that nFIQ interrupt input pins are directly routed to their respective CPU.

Interrupt sources are of the following types:

Interprocessor interrupts (IPI)

Each MP11 CPU has private interrupts, ID0-ID15, that can only be triggered by software. These interrupts are aliased so that there is no requirement for a requesting MP11 CPU to determine its own ID when it deals with IPIs. The priority of an IPI depends on the receiving CPU, not the sending CPU.

Private timer and/or watchdog interrupts.

Each MP11 CPU has its own private timer and watchdog that can generate interrupts, using ID29 and ID30.

A legacy nIRQ pin

In legacy IRQ mode the legacy nIRQ pin, on a per CPU basis, bypasses the Interrupt Distributor logic and directly drives interrupt requests into the MP11 CPU. In legacy IRQ mode, if bit [0] of the CPU Interface Control Register is 0, then no interrupts are raised based on input from the Interrupt Distributor. If bit [0] is 1, then all interrupts are received from the Interrupt Distributor.

When an MP11 CPU uses the Distributed Interrupt Controller (rather than the legacy pin in the legacy mode) by enabling its own CPU interface, the legacy nIRQ pin is treated like other interrupt lines and uses ID31.

Hardware interrupts

Hardware interrupts are triggered by programmable events on associated interrupt input lines. MP11 CPUs can support up to 224 interrupt input lines. The interrupt input lines can be configured to be edge sensitive (posedge) or level sensitive (high level). Hardware interrupts start at ID32.

10.3.1 Interrupt Distributor overview

The Interrupt Distributor holds the list of Pending interrupts for each CPU, and then selects the highest priority interrupt before issuing it to the CPU interface. Interrupts of equal priority are resolved by selecting the lowest ID.

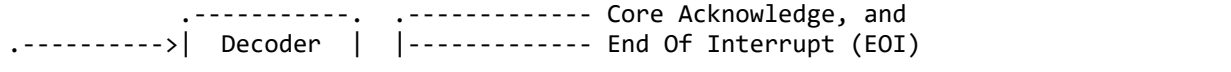
The Interrupt Distributor consists of a register-based list of interrupts, their priorities and activation requirements (CPU targets). In addition the state of each interrupt on each CPU is held in the associated state storage.

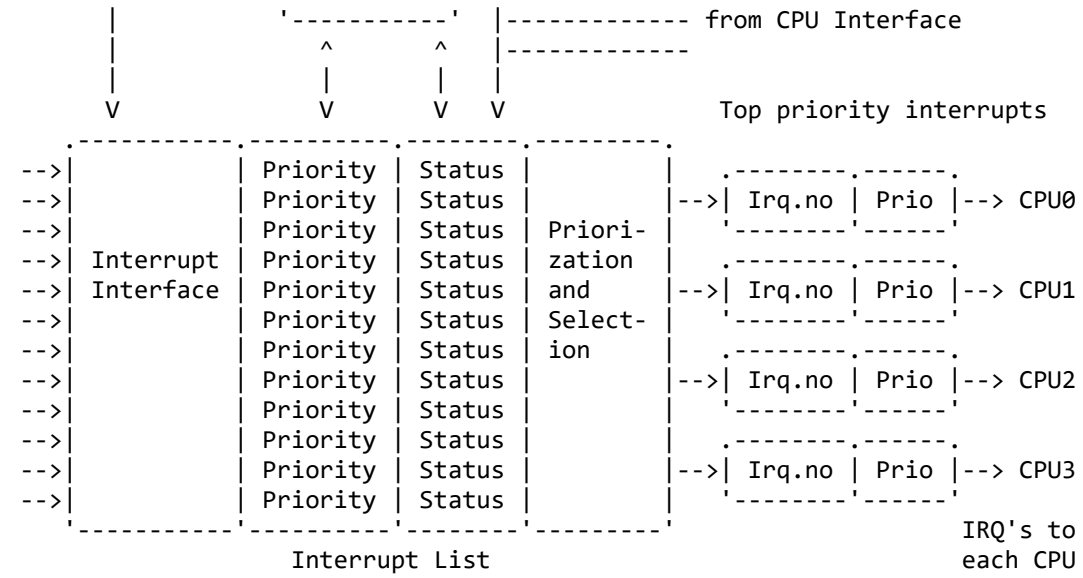
The prioritization logic is physically duplicated for each CPU to enable the selection of the highest priority for each CPU.

The Interrupt Distributor holds the central list of interrupts, processors and activation information, and is responsible for triggering software interrupts to processors.

The CPU Interface acknowledges interrupts and changes interrupt priority masks.

The Interrupt Distributor transmits to the CPU interrupt interfaces their highest Pending interrupt. It receives back the information that the interrupt has been acknowledged, and can then change the status of the corresponding interrupt. The CPU Interface also transmits End of Interrupt Information (EOI), which enables the Interrupt Distributor to update the status of this interrupt from Active to Inactive.





10.3.2 Behavior of the Interrupt Distributor
When the Interrupt Distributor detects an interrupt assertion, it sets the status of the interrupt for the targeted MP11 CPUs to Pending. Level-triggered interrupts cannot be marked as Pending if they are already Active for at least one MP11 CPU.
For each MP11 CPU the prioritization and selection block searches for the Pending interrupt with the highest priority. This interrupt is then sent with its priority to the CPU Interface.

The CPU Interface returns information to the Distributor when the CPU acknowledges (Pending to Active transition) or clears an interrupt (Active to Inactive transition). With the given interrupt ID, the Interrupt Distributor updates the status of this interrupt according to the information sent by the CPU Interface.
When an interrupt is triggered by the Software Interrupt Register or the Set-pending Register, the status of that interrupt for the targeted CPU or CPUs is set to Pending. This interrupt then has the same behavior as a hardware interrupt. The distributor does not differentiate between software and hardware triggered interrupts.

ARM Vector Floating-point Unit (VFP)

The VFP unit exists on 3DS ARM11.

ARM VFP Floating Point Registers
ARM VFP Floating Point Control/Status Registers
ARM VFP Floating Point Opcode Encoding
ARM VFP Floating Point Maths Opcodes
ARM VFP Floating Point Load/Store Opcodes

Floating Point
The floating point hardware is called VFPv2 (Vector Floating-point).
```
ARM DDI 0100I ARM Architecture Reference Manual (for ARMv6 wirh VFPv2)
ARM DDI 0360F ARM11 MPCore r2p0, contains more (mostly useless) VFPv2 info
```
The Fxxxx floating point opcodes are aliases for CP10/CP11 copressor numbers; CP10 used for single, and CP11 for double precision instructions.

ARM VFP Floating Point Registers

Floating point Registers

Registers S0-S31 can contain Single-precision float values, or 32bit Integers (for conversion to/from float format), or a pair of two Single-precision registers can contain one Double-precision float value.

```
  Scalar Bank       Vector bank 1       Vector bank 2       Vector bank 3
  S1:S0    D0       S9:S8    D4         S17:S16  D8         S25:S24  D12
  S2:S3    D1       S11:S10  D5         S19:S18  D9         S27:S26  D13
  S5:S4    D2       S13:S12  D6         S21:S20  D10        S29:S28  D14
  S7:S6    D3       S15:S14  D7         S23:S22  D11        S31:S30  D15
```

The VFP supports "Scalar" and "Vector" modes (and a mixed "Vector/Scalar" mode).

The "Vector" mode can perform simultaneous operations on up to 8 singles, or up to 4 doubles (via Vector LEN and STRIDE selected in FPSCR register).

The registers are organized in "banks", and vectors cannot cross banks (eg. using operand S23 with LEN=3, STRIDE=2 would use registers S23,S17,S19).

Scalar Mode, Fd=Fm <op> Fn

The "Scalar" mode performs operations on 1 single or double. This done in any of the following situations:

```
 - When FPSCR register is set to Vector LEN=1 (and STRIDE=1), or
 - When Destination is S0..S7 or D0..D3 (scalar bank), or
 - When using FCMP comparision opcodes, or
 - When using FCVT or FxxTOxx conversion opcodes, or
 - When using FMxxRxx register transfer opcodes, or
 - When using FLDxx/FSTxx load/store (whereof, FLDM/FSTM can transfer multiple
   registers in vector-like fashion; regardless of LEN/STRIDE settings)
```

Vector Mode, Fd[LEN]=Fm[LEN] <op> Fn[LEN]

The vector mode does merely perform the selected operation on all array elements, this is correct for cases like Vector+Vector addition, but incorrect for Vector*Vector multiplication (to get the final result one must manually compute the sum of the results).

```
 - When FPSCR register is set to Vector LEN=2..8 (and STRIDE=1..2), and
 - When Source and Destination are S8..S31 or D4..D15 (vector banks), and
 - When using FADD, FSUB, FDIV, FCPY, FABS, FNEG, FSQRT, or FxMxx multiply
```

Mixed Mode, Fd[LEN]=Fm <op> Fn[LEN]

This allows to add/multiply/etc. all elements of a vector by a scalar value. This is done when combining vectors operands with the following:

```
 - When Source operand Fm is S0..S7 or D0..D3 (scalar bank), and
 - othersise same conditions as for Vector mode
```

Integer Format (S0..S31 aka I0..I31)

```
  31-0   Integer (signed or unsigned, depending on FxxTOxx opcode)
```

The VFP can't do integer maths, however, one can load/store integer values in S0..S31, and then use the FxxTOxx opcodes to convert integers to/from float format. The integers are always 32bit (no matter if converting Single/Double precision float values).

Single Precision Registers (float1.8.23) (S0..S31)

```
  31     1bit  Sign (0=Positive, 1=Negative)
  30-23  8bit  Exponent (01h..FEh=for 2^(N-7Fh), or 00h/FFh=Special)
  22-0   23bit Fraction (0..7FFFFFh)
```

Double Precision Registers (float1.11.52) (D0..D15)

```
  63     1bit  Sign (0=Positive, 1=Negative)
  62-52  11bit Exponent (001h..7FEh=for 2^(N-3FFh), or 000h/7FFh=Special)
  51-0   52bit Fraction (0..FFFFFFFFFFFFFh)
```

Exponent 01h..FEh (Single) or 001h..7FEh (Double):

```
  Sign * 2^(exponent-7Fh) * (1.fraction)    ;Single
  Sign * 2^(exponent-3FFh) * (1.fraction)   ;Double
```

Exponent 00h (Single) or 000h (Double), aka Small Numbers and Zero:

```
Sign * 2^(-7Eh) * (0.fraction)           ;Single
Sign * 2^(-3FEh) * (0.fraction)          ;Double
The above includes 0 being encoded as fraction=0, the sign bit is ignored
for cases like "compare +/-0", but the sign is used for "divide by +/-0".
Small numbers in 0.fraction format may require extra clock cycles for
counting leading zeroes; unknown if that problem does actually exist on
ARM hardware, however, the "flush to zero" feature (see FPSCR.bit24) can
be used to avoid that issue; 0.fraction will be then replaced by 0.000.
```

Exponent FFh (Single) or 7FFh (Double), aka NaN's and Infinite:

```
fraction=000000h          or 0000000000000h                +/-Infinite
fraction=000001h..3FFFFFh or 0000000000001h..7FFFFFFFFFFFh +/-Signaling NaNs
fraction=400000h          or 8000000000000h                +/-Default NaN
fraction=400000h..7FFFFFh or 8000000000000h..FFFFFFFFFFFFh +/-Quite NaNs
NaNs (Not a Number) can be used for abstract non-numeric expressions; this
isn't useful for normal maths, but may be useful if a database contains
entries like "Weight=UNKNOWN". If so, one may handle the NaN before passing
it to the floating point unit, or otherwise the hardware will either trigger
an exception (Signaling NaNs) or leave the NaN unchanged (Quite NaNs),
eg. "UNKNOWN*2+3 = UNKNOWN", or replace it by Default NaN (if FPSCR.bit25=1).
Different NaNs can be compared using integer comparisons, float comparisions
of NaNs have "unordered" results (even when comparing a NaN with itself).
```

Multiply Note

When multiplying vector*vector, the hardware does merely multiply the components (without computing the sum of the multiply results). To get sum, one could use FMUL and several FADD's with different strides. Or better, for multiple vector*vector multiplications, use FMUL and several FMAC's with source data rearranged as so:

```
FMUL (X,X,X,X,X,X,X,X)*(X,X,X,X,X,X,X,X)
FMAC (Y,Y,Y,Y,Y,Y,Y,Y)*(Y,Y,Y,Y,Y,Y,Y,Y)
FMAC (Z,Z,Z,Z,Z,Z,Z,Z)*(Z,Z,Z,Z,Z,Z,Z,Z)
FMAC (W,W,W,W,W,W,W,W)*(W,W,W,W,W,W,W,W)
```

Ie. in that case, the 1st "vector" contains the X components from up to eight (X,Y,Z,W) vectors.

ARM VFP Floating Point Control/Status Registers

FPSID Register (Floating Point System ID) (R)

```
31-24  Implementor code (41h=ARM)
23     Hardware/software implementation (0=Hardware, 1=Software)
22-21  FSTMX/FLDMX format  (0=Format 1, Other=Reserved)
20     Supported Precision (0=Single and Double, 1=Single only)
19-16  Architecture version number (0=VFPv1, 1=VFPv2, 2-15=Reserved)
15-8   Primary part number of VFP implementation (20h=VFP11) ;\Implementation
7-4    Variant number                            (0Bh=MPCore); defined
3-0    Revision number of the part               (04h=Fourth);/
```

New3DS: 410120b4h = VFPv2 D variant (with single AND double precision).

FPSCR Register (Floating Point Status/Control Register for user-level) (R/W)

```
31     N Flag (1=Comparision result is Less Than)
30     Z Flag (1=Comparision result is Equal)
29     C Flag (1=Comparision result is Equal, Greater Than, or Unordered)
28     V Flag (1=Comparision result is Unordered)
       Note: Use FMSTAT opcode to transfer above flags to ARM CPSR flags
27-26  Unused (0)
25     Default Nan mode   (XXX see page C2-16) (0=Disable, 1=Enable)
24     Flush-to-zero mode (XXX see page C2-14) (0=Disable, 1=Enable)
23-22  Rounding mode (0=To Nearest, 1=Up, 2=Down, 3=Towards Zero)
21-20  Vector Stride (0/3 = 1/2 Singles; or 0/3 = 1/2 Doubles) (1/2=Reserved)
19     Unused (0)
18-16  Vector Len    (0..7 = 1..8 Singles; or 0..3 = 1..4 Doubles)
```

```
  15     Trap Enable Input Denormal (aka Subnormal)     ;\
  14-13  Unused (0)                                     ;
  12     Trap Enable Inexact                            ; Trap Enable aka
  11     Trap Enable Underflow                          ; Exception Enable
  10     Trap Enable Overflow                           ;
  9      Trap Enable Division by Zero                   ;
  8      Trap Enable Invalid Operation                  ;/
  7      Cumulative Exception Input Denormal            ;\
  6-5    RES                                            ;
  4      Cumulative Exception Inexact                   ; Cumulative what...?
  3      Cumulative Exception Underflow                 ;
  2      Cumulative Exception Overflow                  ;
  1      Cumulative Exception Division by Zero          ;
  0      Cumulative Exception Invalid Operation         ;/
```

FPEXC Register (Floating Point Exception Register for system-level) (R/W)

```
  31     Exception Flag ... long blurb replated to process swap code
  30     Enable Floating Point Instructions (0=Disable, 1=Enable)
  29-0   Sub-architecture defined (see below for mpcore)
 Extra mpcore bits:
  29     Unused (0)
  28     FPINST2 instruction valid flag
  27-11  Unused (0)
  10-8   VECITR Number of remaining iterations after exception (0..6=1..7, 7=0)
  7      INV Input exception flag
  6-4    Unused (0)
  3      UFC Potential Underflow Flag
  2      OFC Potential Overflow Flag
  1      Unused (0)
  0      IOC Potential invalid operation flag
```

The exception handler must clear bit31 and bit28.

FPINST - Floating-Point Instruction Register, Privileged 0xEE000A00 (R/W)

Contains the opcode that has triggered the exception. The Cond field in bit28-31 is changed to 0Eh (Always), and the Fd:D, Fn:N, Fm:M are changed to indicated the fault-location within a vector (with FPEXC.bit8-10 indicating the remaining unprocessed elements of the vector).

FPINST2 - Floating-Point Instruction Register 2, Privileged UNP (R/W)

If FPEXC.bit28=1, then this register contains another float opcode (that was prefetched, but not yet executed). The Cond field in bit28-31 is changed to 0Eh (Always). The exception handler should handle the failed FPINST opcode, then try to execute prefetched FPINST2 opcode, and then return from exception.

MVFR0, Media and VFP Feature Register 0, Any 0x11111111 (R)

```
  31-28  VFP hardware support level when user traps are disabled
         (01h=In MPCore processors when Flush-to-Zero and Default_NaN and
         Round-to-Nearest are all selected in FPSCR, the coprocessor does not
         require support code. Otherwise floating-point support code is
         required)
  27-24  Support for short vectors           (01h=Yes)
  23-20  Support for hardware square root    (01h=Yes)
  19-16  Support for hardware divide         (01h=Yes)
  15-12  Support for software/user traps     (01h=Yes/support code is required)
  11-8   Support for double precision VFP    (01h=Yes, v2)
  7-4    Support for single precision VFP    (01h=Yes, v2)
  3-0    Support for the media register bank (01h=Yes/support 16, 64bit regs)
```

MVFR1 - Media and VFP Feature Register 1, Any 0x00000000 (R)

```
  31-28  Reserved
  11-8   Support for media extension, single precision floating-point (00h=No)
  7-4    Support for media extension, integer instructions            (00h=No)
  3-0    Support for media extension, load/store instructions         (00h=No)
```

ARM VFP Floating Point Opcode Encoding

Comparision of normal ARM copro opcodes and VFP opcodes

```
  |..3 .................2 .................1 .................0|
  |1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0|
  |_Cond__|1_1_0_0_0_1_0|L|__Rn___|__Rd___|__CP#__|_CPopc_|__CRm__| 2reg normal
  |_Cond__|1_1_0_0_0_1_0|L|__Rn___|__Rd___|__CP#__|0|0|M|1|__Fm___| 2reg on VFP
  |_Cond__|1_1_0|P|U|N|W|L|__Rn___|__CRd__|__CP#__|____Offset_____| Mem normal
  |_Cond__|1_1_0|P|U|D|W|L|__Rn___|__Fd___|__CP#__|____Offset_____| Mem on VFP
  |_Cond__|1_1_1_0|_CPopc_|__CRn__|__CRd__|__CP#__|_CP__|0|__CRm__| CDP normal
  |_Cond__|1_1_1_0|p|D|q|r|__Fn___|__Fd___|__CP#__|N|s|M|0|__Fm___| CDP on VFP
  |_Cond__|1_1_1_0|CPopc|L|__CRn__|__Rd___|__CP#__|_CP__|1|__CRm__| 1reg normal
  |_Cond__|1_1_1_0|CPopc|L|__Fn___|__Rd___|__CP#__|N|0_0|1|0_0_0_0| 1reg on VFP

  Cond             = Condition
  L                = Load/Store direction for memory/register transfers
  Fm:M, Fn:N, Fd:D = Float Registers S0..S31 (or D0..D15, with LSB=0)
  Rd, Rn           = ARM Registers
  PUW, pqrs, CPopc = Opcode bits
  CP#              = Coprocessor number (0Ah=Single-, 0Bh=Double-Precision)
  Offset           = Address step, implies number of registers for FLDM/FSTM
```

ARM VFP Floating Point Maths Opcodes

VFP data-processing primary opcodes

```
  pqrs cp10/cp11                  Instruction functionality
  0000 FMAC{S|D}{cond} Fd,Fn,Fm   Fd = +(Fn*Fm)+Fd   ;Multiply, Add
  0001 FNMAC{S|D}{cond} Fd,Fn,Fm  Fd = -(Fn*Fm)+Fd   ;Multiply, Negate, Add
  0010 FMSC{S|D}{cond} Fd,Fn,Fm   Fd = +(Fn*Fm)-Fd   ;Multiply, Subtract
  0011 FNMSC{S|D}{cond} Fd,Fn,Fm  Fd = -(Fn*Fm)-Fd   ;Multiply, Negate, Sub
  0100 FMUL{S|D}{cond} Fd,Fn,Fm   Fd = +(Fn*Fm)      ;Multiply
  0101 FNMUL{S|D}{cond} Fd,Fn,Fm  Fd = -(Fn*Fm)      ;Multiply, Negate
  0110 FADD{S|D}{cond} Fd,Fn,Fm   Fd = Fn+Fm         ;Add
  0111 FSUB{S|D}{cond} Fd,Fn,Fm   Fd = Fn-Fm         ;Sub
  1000 FDIV{S|D}{cond} Fd,Fn,Fm   Fd = Fn/Fm         ;Divide
  1001 -Undefined-
  1010 -Undefined-
  1011 -Undefined-
  1100 -Undefined-
  1101 -Undefined-
  1110 -Undefined-
  1111 -Extension instructions-
```

VFP data-processing extension opcodes

```
  Fn   N cp10/cp11                  Instruction functionality
  0000 0 FCPY{S|D}{cond} Fd,Fm      Fd = Fm        ;Copy
  0000 1 FABS{S|D}{cond} Fd,Fm      Fd = abs(Fm)   ;Absolute
  0001 0 FNEG{S|D}{cond} Fd,Fm      Fd = -Fm       ;Negate
  0001 1 FSQRT{S|D}{cond} Fd,Fm     Fd = sqrt(Fm)  ;Square root
  001x x -Undefined-
  0100 0 FCMP{S|D}{cond} Fd,Fm      Fd-Fm     ;Compare
  0100 1 FCMPE{S|D}{cond} Fd,Fm     Fd-Fm     ;Compare, exception on quiet NaNs
  0101 0 FCMPZ{S|D}{cond} Fd        Fd-0      ;Compare
  0101 1 FCMPEZ{S|D}{cond} Fd       Fd-0      ;Compare, exception on quiet NaNs
  0110 x -Undefined-
  0111 0 -Undefined-
  0111 1 FCVT{DS|SD}{cond} Fd,Fm    Single <--> Double-precision conversion
  1000 0 FUITO{S|D}{cond} Fd,Im     Unsigned integer --> float
```

```
  1000 1 FSITO{S|D}{cond} Fd,Im     Signed integer --> float
  1001 x -Undefined-
  101x x -Undefined-
  1100 0 FTOUI{S|D}{cond} Id,Fm     Float --> unsigned integer
  1100 1 FTOUIZ{S|D}{cond} Id,Fm    Float --> unsigned integer, round to zero
  1101 0 FTOSI{S|D}{cond} Id,Fm     Float --> signed integer
  1101 1 FTOSIZ{S|D}{cond} Id,Fm    Float --> signed integer, round to zero
  111x x -Undefined-
```
cp10: FCVTDS Dd,Sm ;Double <-- Single
cp11: FCVTSD Sd,Dm ;Single <-- Double

Nocash syntax
The useless {S|D} and {DS|SD} suffixes are ommitted. FCVT is renamed to FMOV. F{UI|SI}TO{UI|SI}{Z} is renamed to FMOV{UI|SI}{Z}, with operand I0..I31 for the integer register.

ARM VFP Floating Point Load/Store Opcodes

VFP single register transfer instructions
```
  cp   opcode L Instruction name   Instruction functionality
  cp10 000    0 FMSR{cond} Sn,Rd   Sn = Rd        ;\Single-Precision or Integer
  cp10 000    1 FMRS{cond} Rd,Sn   Rd = Sn        ;/
  cp10 111    0 FMXR{cond} sys,Rd  Reg(Fn,N) = Rd ;\SystemReg (FPSID, etc.)
  cp10 111    1 FMRX{cond} Rd,sys  Rd = Reg(Fn,N) ;/  ;<-- or FMSTAT{cond}
  cp11 000    0 FMDLR{cond} Dn,Rd  Dn.31-0 = Rd   ;\LSW of Double-Precision
  cp11 000    1 FMRDL{cond} Rd,Dn  Rd = Dn.31-0   ;/
  cp11 001    0 FMDHR{cond} Dn,Rd  Dn.63-32 = Rd  ;\MSW of Double-Precision
  cp11 001    1 FMRDH{cond} Rd,Dn  Rd = Dn.63-32  ;/
  other's       -Undefined-
```
System Register encodings:
```
  Fn   N  System register
  0000 0  FPSID (New3DS: 410120b4h = VFPv2 with single AND double precision)
  0001 0  FPSCR      ;(FMSTAT opcode encodes as FMRX R15,FPSCR)
  0110 0? MVFR1                     ;\mpcore only
  0111 0? MVFR0                     ;/
  1000 0  FPEXC
  1001 0? FPINST                    ;\mpcore only
  1010 0? FPINST2                   ;/
```

VFP two register transfer instructions (VFPv2 and above)
```
  cp   L Instruction name              Instruction functionality
  cp10 0 FMSRR{cond} {Sm,Sm+1},Rd,Rn   Fm = Rn, (Fm+1) = Rd    ;XXX swapped?
  cp10 1 FMRRS{cond} Rd,Rn,{Sm,Sm+1}   Rn = Fm, Rd = (Fm+1)    ;XXX swapped?
  cp11 0 FMDRR{cond} Dm,Rd,Rn          Fm[31:0] = Rd, Fm[63:32] = Rn
  cp11 1 FMRRD{cond} Rd,Rn,Dm          Rd = Fm[31:0], Rn = Fm[63:32]
```

VFP load and store instructions
```
  PUW  L=0/1,cp10/cp11                            Registers transferred
  000  -Two-register transfer instructions-       -
  001  -Undefined-                                -
  010  FSTM|FLDMIA{S|D|X}{<cond>} Rn,{Fd,Fd+1,..}  Multiple Registers
  011  FSTM|FLDMIA{S|D|X}{<cond>} Rn!,{Fd,Fd+1,..} Multiple Registers Increment
  100  FST|FLD{S|D}{<cond>} Fd, [Rn{,-offs*4}]     One register, -offs
  101  FSTM|FLDMDB{S|D|X}{<cond>} Rn!,{Fd,Fd+1,..} Multiple Registers Decrement
  110  FST|FLD{S|D}{<cond>} Fd, [Rn{,+offs*4}]     One register, +offs
  111  -Undefined-                                -
```
FSTM/FLDM do transfer multiple words (with offs containing the number of words to be transferred, 1..32 for {S}, or an even number 2..32 for {D}.
VFP load/store multiple addressing modes
```
  Non-stacking mnemonic   Stacking mnemonic
```

```
FLDMIA{S|D|X}          FLDMFD{S|D|X}     FPOP{S|D|X}
FLDMDB{S|D|X}          FLDMEA{S|D|X}
FSTMIA{S|D|X}          FSTMEA{S|D|X}
FSTMDB{S|D|X}          FSTMFD{S|D|X}     FPUSH{S|D|X}
```

Nocash syntax
The useless {S|D} suffixes are ommitted, the weird {X} suffix is kept used to preserve weirdness.
Fancy {} brackets are omitted, LDM/STM must use square [Rn] brackets, the register list in LDM/STM is specified as Fx-Fz (rather than Fx,Fy,Fz).
All FMxxxx opcodes are renamed to FMOV{LSW|MSW}.

Weird STM/LDM{X} - for registers with unknown precision
The weird {X} mode is same as {D}, but with offset.bit0=1 (ie. with offs=3..33 instead of 2..32; and thereby actually transferring an unused dummy word).
The weird {X} mode is/was intended for registers with unknown content (eg. when pushing/popping registers without knowing if they contain integer/single/double precision values; which might be a problem with internal accumulators in the VFP unit).
The weird {X} mode was declared as "deprecated in ARMv6" in DDI 0100I, but later re-declared as required for "compatibility with future VFP implementations" in DDI 360F. However, unknown if there are/were/will be any such implementations that do require it.
For now, it should be best to use {D} mode instead of weird {X}. Probably even {S} should also work the same (if the endianess-based word-order doesn't matter).

3DS NCSD Format

There are two known specialisations of the NCSD container format:
- The CTR Cart Image (CCI) format (CCI is the format of game ROM images)
- The 3DS' raw NAND format

CTR System Update (CSU) is a variant of CCI, where the only difference is in the file extension (uh, that is the file extension of a ROM-image that needs to be copied to a dev flashcard or so?). This is used with developer System Updates and associated Tools.

NCSD images start with a NCSD header, followed by up to a maximum of 8 NCCH partitions (NCCH... uh, yeah? that's not so for eMMC, but maybe applies to ROM carts?).

For CCI images, the partitions are reserved as follows:

```
NCCH Index    Reserved Use
0             Executable Content (CXI)
1             E-Manual (CFA)
2             Download Play Child container (CFA)
6             New3DS Update Data (CFA)   ;\aka firmware updates?
7             Old3DS? Update Data (CFA)  ;/
```

The format of partitions can be determined from the partition FS flags (normally these are zero for CCI/CSU NCSD Images) (uh, what/where is "FS flags"?).

NCSD header

```
000h 100h    RSA-2048 SHA-256 signature of the NCSD header (including MBR!)
100h 4       ID "NCSD"
104h 4       Size of the NCSD image, in media units
               (1 media unit = 200h bytes)  (aka what is called "LBA" ?)
108h 8  ?    Media ID
110h 1*8     Partitions Type  (0=Unused, 1=MBR, 3=FIRM, 4=GBA-SAVE)
118h 1*8     Partitions Crypt (0=Unused, 1=DSi, 2=3DS, 3=New3DS)
120h (4+4)*8 Partitions Offset & Length (in media units)
For carts:
```

```
  160h 20h Exheader SHA-256 hash
  180h 4   Additional header size (uh, does that mean "Exheader"?)
  184h 4   Sector zero offset
  188h ..  Partition Flags (uh, why called "flags" and why "partition"?)
  188h 1   Backup Write Wait Time (The time to wait to write save to backup
             after the card is recognized (0-255 seconds)). NATIVE_FIRM loads
             this flag from the gamecard NCSD header starting with 6.0.0-11.
  189h 1   unknown, maybe related to below "Save Crypto"?
  18Ah 1   unknown, ?
  18Bh 1   Media Card Device (1=NOR Flash, 2=None, 3=BT) (SDK 3.X+)
  18Ch 1   Media Platform Index (1=CTR)
  18Dh 1   Media Type Index (0=InnerDevice, 1=Card1, 2=Card2, 3=ExtendedDevice)
  18Eh 1   Media Unit Size (200h SHL N)
  18Fh 1   Media Card Device (1=NOR Flash, 2=None, 3=BT) (Only SDK 2.X)
  190h 40h Partition ID table (8x8 bytes)
  1D0h 20h Reserved
  1F0h 0Eh Reserved?
  1FEh 1   Support for this was implemented with 9.6.0-X FIRM.
             Bit0=1 enables using bits 1-2, it's unknown what these two
             bits are actually used for (the value of these two bits get
             compared with some other value during NCSD
             verification/loading).
             This appears to enable a new, likely hardware-based,
             antipiracy check on cartridges.
  1FFh 1   Support for this was implemented with 9.6.0-X FIRM, see below
             regarding save crypto.
 For eMMC:
  160h 25h Zerofilled
  185h 1   Unknown (04h)
  186h 6   Zerofilled
  18Ch 1   Unknown (01h)
  18Dh 1   Zero
  18Eh 1   Shift amount for blocks vs 200h-byte sectors (00h)
  18Fh 2Fh Zerofilled (unencrypted, despite of below)
  1BEh 42h Encrypted MBR partition-table, for the TWL partitions
             (key-data used for this keyslot is console-unique).
```

NCSD Signature

The RSA pubk used for gamecard NCSD is stored in ITCM.

The RSA pubk used for NAND NCSD is stored in "Process9 .(ro)data instead of ITCM". Uh, shouldn't that key (also?) be in bootrom?

Partition Flags (In Terms of Save Crypto Determination) (uh, Save what?)

```
  Byte   Description
  01h    Starting with 6.0.0-11 NATIVE_FIRM will use this flag to determine
           the gamecard savegame keyY method, when flag[3] is set.
           00h = 2.0.0-2 hashed keyY,
           01h = new keyY method implemented with 6.0.0-11.
           0Ah = implemented with 9.3.0-X. On Old3DS this is identical to
             the 2.2.0-4 crypto. On New3DS this is identical to the 2.2.0-4
             crypto, except with New3DS-only gamecard savedata keyslots.
```

Starting with 9.6.0-X FIRM, Process9 now sets <savecrypto_stateval> to partitionflag[1] + <the u8 value from NCSD+0x1FF>, instead of just setting it to partitionflag[1].

```
  03h    Support for this flag was implemented in NATIVE_FIRM with 2.0.0-2.
           When this flag is set the hashed gamecard savegame keyY method is
           used, this likely still uses the repeating-CTR however.
           With 6.0.0-11 the system will determine the gamecard savegame keyY
           method via flag[1], instead of just using the hashed keyY via this
           flag.
  07h    This flag enables using the hashed gamecard savegame keyY method,
           support for this flag was implemented in NATIVE_FIRM with 2.2.0-4.
           All games with the NCSD image finalized since 2.2.0-4 (and contains
```

```
          2.2.0-4+ in the system update partition) have this flag set, this
          flag also enables using new CTR method as well.
```
Starting with 9.6.0-X FIRM, Process9 will just write val0 to a state field then return 0, instead of returning an error when the save crypto type isn't recognized. This was the *only* actual functionality change in the Old3DS Process9 function for gamecard savedata crypto init.

Card Info Header - uh, maybe this is the "Exheader" aka "Additional header"?
```
  Offset Size  Description
  200h   4     CARD2: Writable Address In Media Units (For 'On-Chip' Savedata)
               CARD1: Always FFFFFFFFh.
  204h   4     Card Info Bitmask
  208h   108h  Reserved1
  310h   2     Title version
  312h   2     Card revision
  208h   CEEh  Reserved2
  1000h  10h   Card seed AES-KeyY (first u64 is Media ID (same as first NCCH
                 partitionId))
  1010h  10h   Encrypted card seed (AES-CCM, keyslot 3Bh for retail cards,
                 see CTRCARD_SECSEED)
  1020h  10h   Card seed AES-MAC
  1030h  0Ch   Card seed AES-IV
  103Ch  C4h   Reserved3
  1100h  100h  Copy of first NCCH header (excluding RSA signature)
 Development Card Info Header Extension:
  1200h  200h  CardDeviceReserved1
  1400h  10h   TitleKey
  1410h  F0h   CardDeviceReserved2
```
Note that a particular(=?) flashcard vendor puts what many refer to as "private headers" here in place of actual development card information. This header is constituted(=?) by a cartridge-unique Id obtained from pxi:ps9::GetRomId and the title-unique cart ID (identical for all carts of the same title; can be retrieved using the NTR gamecard protocol command 90h or through the CTR protocol commands 90h or A2h).

Tools
ctrtool - (CMD)(Windows/Linux) Parsing NCSD files
3DSExplorer - (GUI) (Windows Only) Parsing NCSD files

3DS FIRM Format

File format for the 3DS' Firmware, it contains up to four 'sections' of data comprising the ARM9 and ARM11 kernels, and some fundamental processes.
The firmware sections are not encrypted.

The ARM9 section contains the ARM9 kernel (and loader) and the Process9 NCCH (which is the only process run in user mode on the ARM9). The ARM11 sections contain the ARM11 kernel (and loader), and various ARM11 process NCCHs. For NATIVE_FIRM/SAFE_MODE_FIRM these ARM11 processes are sm, fs, pm, loader, and pxi. Normally the 4th section is not used. The code loaded from FIRM is constantly running on the system until another FIRM is launched. The ARM11 kernel is hard-coded to always decompress the ExeFS .code of embedded ARM11 NCCHs without checking the exheader compression bit.

FIRM Format can exist in two locations:
```
  As raw eMMC sectors (defined in the NCSD Header of the eMMC storage)
  As .firm files (in NCCH .app files, in the "\title\00040138" firmware folder)
```
And, in Wifi-Flash and NTRCARDs (that's both only for repair/unbricking).

FIRM Header
```
  000h  4       ID "FIRM"
  004h  4       Boot priority (0=Normal/Lowest)
```

```
  008h  4       ARM11 Entrypoint (usually 1FFxxxxxh, aka AXI WRAM.. or DSP?)
  00Ch  4       ARM9 Entrypoint  (usually 080xxxxxh, aka ARM9-only RAM)
  010h  30h     Reserved (0)
  040h  30h     Firmware Section Header 1 (usually several NCCH's with ExeFS)
  070h  30h     Firmware Section Header 2 (usually ARM11 code)
  0A0h  30h     Firmware Section Header 3 (usually ARM9 code)
  0D0h  30h     Firmware Section Header 4 (often unused, zerofilled)
  100h  100h    RSA-2048 SHA-256 signature of the FIRM header
```
The RSA signature is checked when bootrom/Process9 are doing FIRM-launch (with the public key being hardcoded in each). The signature is not checked when installing FIRM to the NAND firm0/firm1 partitions.

Firmware Section Headers
```
  000h  4       Byte offset      (relative to begin of FIRM header)
  004h  4       Physical address where the section is loaded to
  008h  4       Byte-size    (or 0=None)
  00Ch  4       Copy-method (0=NDMA, 1=XDMA, 2=CPU mem-copy)
                            (0=ARM9/WRAM, 1=ARM11/AXI/VRAM, 2=?)
  010h  20h     SHA-256 Hash of Firmware Section
```
Note on Copy-method: Process9 ignores this field. Boot9 doesn't immediately throw an error when this isn't 0..2. In that case it will jump over section-data-loading which then results in the hash verification with the below hash being done with the hash already stored in the SHA hardware.

FIRM Load Areas
Common Load areas are:
```
  08006000h 0.9M  ARM9-WRAM  (common for ARM9 code) (not first/last some bytes)
  18000000h 6.0M  VRAM       (eg. used by GBA/DSi firmwares)
  1FF00000h 0.5M  DSP Memory (eg. used by FIRM0)
  1FF80000h 0.5M  AXI RAM    (common for ARM11 code)
```
With that areas, a FIRM file can be max 7.9Mbyte (if it is loaded as file, the default size of the firm0/firm1 partitions is only 4Mbyte each).
There is a blacklist in bootrom preventing some memory regions:
```
  07FFB800h..07FFFBFFh  ARM9 ITCM part (otp, mbr, keys...)
  FFF00000h..FFF02FFFh  ARM9 DTCM (first 3000h) (arm9 data)
  FFF03000h..FFF03FFFh  ARM9 DTCM (last 1000h) (arm9 stack)
  080F8000h..080FFFFFh  ARM9 WRAM (last 8000h) (rom card related?)
  08000000h..0800003Fh  ARM9 WRAM (first 40h) (exception vectors, etc)
  20000000h..27FFFFFFh  FCRAM (whole 128MB)
  FFF00000h..1FFFFFFFh  Bugged (size of that area is negative/nonsense)
```
Moreover, the bootrom doesn't have Main RAM and New3DS memory enabled. And, the bootrom PU has also disabled some memory areas (though those might pass when using DMA for loading?).

3DS FIRM Encryption

New_3DS FIRM
For New3DS firmwares (NATIVE_FIRM, TWL_FIRM, ..), the ARM9 FIRM binary has an additional layer of crypto. At the end of each ARM9 binary, there's a plaintext loader. The format of the FIRM header is identical to regular 3DS FIRM (the RSA modulus is the same as regular 3DS too).

Before checking CFG9_SYSPROT9 the loader main() does the following:
On 9.5.0-X: executes a nop instruction with r0=0 and r1=<address of arm9binhdr+0x50>.
Clears bit6 in REG_AES_KEYCNT (=use 3DS mode for key X/Y instead DSi mode).

If CFG9_SYSPROT9 bit 1 is clear (which means the OTP area is unlocked and so it knows that this is a hard reboot), it does the following things:
Clears 200h-bytes on the stack, then reads NAND Sector 96h (NAND image offset 12C00h), with size 200h-

bytes into that stack buffer.
Checks CFG9_SYSPROT9 bit 1 again, if it's set then it executes a panic function (set r0-r2=0, execute nop instruction, then execute instruction "bkpt 0x99").
Hashes data from the OTP region 0x10012000-0x10012090 using SHA256 via the SHA hardware.
Clears bit6 in REG_AES_KEYCNT. Initializes AES keyslot 0x11 keyX, keyY to the lower and higher portion of the above hash, respectively (uh, where is "lower" in a big-endian SHA value?). Due to the above hashed data, the keyX+keyY here are console-unique.
Decrypts the first 0x10-byte block in the above read NAND sector with keyslot 0x11 using AES-ECB. 9.6.0-X: Then it decrypts the 0x10-bytes at offset 0x10 in the sector with keyslot 0x11.
Then the normalkey, keyX, and keyY, for keyslot 0x11 are cleared to zero. Runs the TWL key-init/etc code which was originally in the ARM9-kernel, then writes 0x2 to CFG9_SYSPROT9 to disable the OTP area.
Then it uses the above decrypted block from sector+0 to set the normalkey for keyslot 0x11. Decrypts arm9_bin_buf+0 using keyslot 0x11 with AES-ECB, and initialises keyX for keyslot 0x15 with it.
9.6.0-X: Then it uses the above decrypted block from sector+0 to set the normalkey for keyslot 0x11. Decrypts a 0x10-byte block from arm9loader .(ro)data using keyslot 0x11 with AES-ECB, and initializes keyX for keyslot 0x18 with it (same block as previous versions).
9.6.0-X: Starting with this version keyslot 0x16 keyX init was moved here, see below for details on this. The code for this is same as 9.5.0-X, except the decrypted normalkey from sector+0x10 is used for keyslot 0x11 instead.
Initialises KeyX for keyslots 0x18..0x1F (0x19..0x1F with 9.6.0-X) with the output of decrypting a 0x10-byte block with AES-ECB using keyslot 0x11. This block was changed to a new one separate from keyslot 0x18, starting with 9.6.0-X. The last byte in this 0x10-byte input block is increased by 0x01 after initializing each keyslot. Before doing the crypto each time, the loader sets the normal-key for keyslot 0x11 to the plaintext normalkey from sector+0(+0x10 with 9.6.0-X). These are New3DS-specific keys.
9.5.0-X (moved to above with 9.6.0-X): Sets the normal-key for keyslot 0x11 to the same one already decrypted on the stack. Decrypts the 0x10-byte block at arm9binhdr+0x60 with AES-ECB using keyslot 0x11, then sets the keyX for keyslot 0x16 to the output data.
9.5.0-X: The normalkey, keyX, and keyY, for keyslot 0x11 are then cleared to zero.

When CFG9_SYSPROT9 bit 1 is set (which means this happens only when this loader runs again for firm-launch), the normalkey, keyX, and keyY, for keyslot 0x11 are cleared to zero.

It sets KeyY for keyslot 0x15 (0x16 with 9.5.0-X) to arm9_bin_buf+16, the CTR to arm9_bin_buf+32 (both are unique for every version). It then proceeds to decrypt the binary with AES-CTR. When done, it sets the normal-key for the keyslot used for binary decryption to zeros. It then decrypts arm9_bin_buf+64 using an hardcoded keyY for keyslot 0x15 (9.5.0-X/9.6.0-X also uses keyslot 0x15), sets the normal-key for this keyslot to zeros again, then makes sure the output block is all zeroes. If it is, it does some cleanup then it jumps to the entrypoint for the decrypted binary. Otherwise it will clear the keyX, keyY, and normal-key for each of the keyslots initialized by this loader (on 9.6.0-X+, on older versions this was bugged and cleared keys 0x00..0x07 instead of 0x18..0x1F), do cleanup (same cleanup as when the decrypted block is all-zero) then just loop forever.

Thus, the ARM9 binary has the following header:

```
000h 16  Encrypted KeyX (same for all FIRM's)
010h 16  KeyY
020h 16  CTR       ;uh, is that the IV value?
030h 8   Size of encrypted binary, as ASCII text?
038h 8   ?
040h 16  Control block
050h 16  Added with 9.5.0-X. Only used for hardware debugging: a nop
           instruction is executed with r0=0 and r1=<address of this
           data>.
060h 16  Added with 9.5.0-X. Encrypted keyX for keyslot 0x16.
```

Originally the padding after the header before offset 0x800 (start of actual ARM9-binary) was 0xFF bytes, with 9.5.0-X this was changed to 0x0.

For the New3DS NATIVE_FIRM arm9-section header, the only difference between the 8.1.0-0_New3DS version and the 9.0.0-20 version is that the keyY, CTR, and the block at 0x30 in the header were updated.

New3DS ARM9 binary loader versions

FIRM system version(s)	Description
8.1.0-0_New3DS - 9.3.0-X	Initial version.
9.5.0-X	See note
9.6.0-X - 11.3.0-X	See above and here.

For 9.5.0-X: Added keyX initialization for keyslot 0x16 (see above), and added code for clearing keyslot 0x11 immediately after the code finishes using keyslot 0x11. The keyslot used for arm9bin decryption was changed from 0x15 to 0x16. Added code for clearing keyslot 0x16 when control-block decryption fails. Added code for using arm9bin_hdr+0x50 with a nop instruction, at the very beginning of the main arm9-loader function. Added two new 0x10-blocks to the arm9bin-hdr.

3DS FIRM Versions

New3DS ARM9 kernel

The only actual code-difference for the Old3DS/New3DS ARM9-kernels' crt0, besides TWL AES / 0x10012000 related code, is that the New3DS ARM9-kernel writes 0x1 to REG_EXTMEMCNT9 in the crt0.

New3DS Process9

The following is all of the differences for Old3DS/New3DS Process9 with 9.3.0-X:
The FIRM-launch code called at the end of the New3DS proc9 main() has different mem-range checks.
In the New3DS proc9, the v6.0/v7.0 keyinit function at the very beginning (before the original code) had additional code added for setting CTRNAND keyslot 0x5, with keydata from .data. After setting the keyY, the keyY in .data is cleared.
In New3DS proc9, the functions for getting the gamecard crypto keyslots / NCCH keyslot can return New3DS keyslots when New3DS flags (NCSD/NCCH) are set.
The code/data for the binary near the end of arm9mem is slightly different, because of memory-region sizes.
The only difference in .data (besides the above code binary) is that the New3DS proc9 has an additional 0x10-byte block for the keyslot 0x5 keyY, see above.

Variations

There exists different official firmwares for the 3DS: The default one (NATIVE_FIRM) is used to run all 3DS content and boots by default, while backwards compatibility is handled by TWL_FIRM and AGB_FIRM. There furthermore is a rescue mode provided by SAFE_MODE_FIRM.

NATIVE_FIRM

NATIVE_FIRM is the FIRM which is installed to the NAND firm partitions, which is loaded by bootrom.
Version history:

System version	old 3DS title version	old 3DS hex title contentID	Kernel/FIRM version (old 3DS/new 3DS)	FIRM ARM11-sysmodule Product Code
Factory v0	00	2.3-0	-	(Factory FIRM (titleID 00040001-00000002)
Pre-1.0 -	-	2.23-X	-	(Referenced in the v1.0 Home Menu NCCH plain-region.)
1.0.0 v432	00	2.27-0	-	
1.1.0 v1472	02	2.28-0	-	
2.0.0 v2516	09	2.29-7	-	
2.1.0 v3553	0B	2.30-18	0608builder	

```
  2.2.0   v4595   0F      2.31-40 0909builder
  3.0.0   v5647   18      2.32-15 1128builder
  4.0.0   v6677   1D      2.33-4  0406builder
  4.1.0   v7712   1F      2.34-0  0508builder
  5.0.0   v8758   25      2.35-6  0228builder
  5.1.0   v9792   26      2.36-0  0401builder
  6.0.0   v10833  29      2.37-0  0520builder
  6.1.0   v11872  2A      2.38-0  0625builder
  7.0.0   v12916  2E      2.39-4  1125builder
  7.2.0   v13956  30      2.40-0  0404builder
  8.0.0   v15047  37      2.44-6  0701builder
  8.1.0-0_New3DS N/A N/A  2.45-5  -
  9.0.0   v17120  38      2.46-0  0828builder
  9.3.0   v18182  3F      2.48-3  1125builder
  9.5.0   v19216  40      2.49-0  0126builder
  9.6.0   v20262  49      2.50-1  0311builder
  10.0.0  v21288  4B      2.50-7  0812builder
  10.2.0  v22313  4C      2.50-9  1009builder
  10.4.0  v23341  50      2.50-11 1224builder
  11.0.0  v24368  52      2.51-0  0406builder
  11.1.0  v25396  56      2.51-2  0805builder
  11.2.0  v26432  58      2.52-0  1015builder
  11.3.0  v27476  5C      2.53-0  0126builder
  11.4.0  v28512  5E      2.54-0  0314builder
  11.8.0  v29557  64      2.55-0  0710pseg-ciuser
```

The above kernel/FIRM versions are in the format: <KERNEL_VERSIONMAJOR>.<KERNEL_VERSIONMINOR>-<KERNEL_VERSIONREVISION>.

SAFE_MODE_FIRM

SAFE_MODE is used for running the System Updater. SAFE_MODE_FIRM and NATIVE_FIRM for the initial versions are exactly the same, except for the system core version fields.

TWL_FIRM

TWL_FIRM handles DS(i) backwards compatibility.
The 3DS-mode ARM9 core seems to switch into DSi-mode (for running DSi-mode ARM9 code) by writing to a PDN register (this changes the memory layout to DSi-mode / etc, therefore this register poke *must* be executed from ITCM). This is the final 3DS-mode register poke before the ARM9 switches into DSi-mode. DS(i)-mode ARM7 code is run on the internal ARM7 core, which is started up during TWL_FIRM boot. Trying to read from the exception-vector region (address 0x0) under this DSi-mode ARM7 seems to only return 0x00/0xFF data. Also note that this DSi-mode ARM7 runs code (stored in TWL_FIRM) which pokes some DSi-mode registers that on the DSi were used for disabling access to the DSi bootROMs, however these registers do not affect the 3DS DSi-mode ARM9/ARM7 "bootrom" region (exceptionvector region + 0x8000) at all.

For shutting down the system, TWL_FIRM writes u8 value 8 to I2C MCU register 0x20. For returning to 3DS-mode, TWL_FIRM writes value 4 to that MCU register to trigger a hardware system reboot.

The TWL_FIRM ARM11-process includes a TWL bootloader, see here and here for details.

TWL_FIRM verifies all TWL RSA padding with the following. This is different from the DSi "BIOS" code.
The first byte must be 0x0.
The second byte must be 0x1 or 0x2.
Executes a while(<value of byte at current pos in RSA message>). When the second_byte in the message is 0x1, the byte at curpos must be 0xFF (otherwise the non-zero value of the byte at curpos doesn't matter). This loop must find a zero byte before offset 0x7F in the message otherwise an error is returned.
Returns an address for msg_curpos+1.

totalhashdatasize = rsasig_bytesize - above position in the message for the hashdata. The actual "totalhashdatasize" in the RSA message must be <= <expected hashdata_size>(0x74 for bootloader). Uh, <=<>

()??? The TWL_FIRM code copies the RSA "hashdata" to the output buffer, using the actual size of the RSA "hashdata".

AGB_FIRM
AGB_FIRM handles running GBA VC titles. The ARM9 FIRM section for TWL_FIRM and AGB_FIRM are exactly the same (for TWL_FIRM and AGB_FIRM versions which were updated with the same system-update).

3DS FIRM Launch Parameters

FIRM Launch Parameters
The FIRM-launch parameters structure is located at FCRAM+0, size 0x1000-bytes. The ARM11-kernel copies this structure elsewhere, then clears the 0x1000-bytes at FCRAM+0. It will not handle an existing structure at FCRAM+0 if CFG_BOOTENV is zero. The ARM9 kernel writes some values about the boot environment to AXI WRAM during init to enable this.
Note: it seems NATIVE_FIRM ARM11-kernel didn't parse this during boot until 3.0.0-X?

```
  000h 300h unknown/unspecified (probably as on DSi?)
  300h 100h 'TLNC' block created by TWL applications, handled by NS for
              backwards-compatibility purposes. See here for more info
  400h 4    Flags
  404h 0Ch  unknown/unspecified
  410h 0Ch  This is used for overriding the FIRM_* fields in
              Configuration_Memory, when the flag listed below is set,
              in the following order (basically just data-copy from here
              to 0x1FF80060): "FIRM_?", FIRM_VERSIONREVISION,
              FIRM_VERSIONMINOR, FIRM_VERSIONMAJOR, FIRM_SYSCOREVER,
              and FIRM_CTRSDKVERSION
  41Ch ..   unknown/unspecified
  438h 4    The kernel checks this field for value 0xFFFF, if it matches
              the kernel uses the rest of these parameter fields,
              otherwise FIRM-launch parameters fields are ignored by
              the kernel
  43Ch 4    CRC32 across [400h..53Fh] with [43Ch]=zero
              When invalid the kernel clears the entire buffer used for
              storing the FIRM-params, therefore no actual FIRM-params are
              handled after that
  440h 10h  Titleinfo Program Info, used by NS during NS startup, to
              launch the specified title when the below flag is set
  450h 10h  Titleinfo Program Info. This might be used for returning
              to the specified title, once the above launched title
              terminates?
  460h 4    Bit0: 0 = titleinfo structure isn't set,
                  1 = titleinfo structure is set
  464h ..   unknown/unspecified
  480h 20h  Can be set via buf1 for APT:SendDeliverArg/APT:StartApplication
  4A0h 10h  Can be set by NSS:SetWirelessRebootInfo
  4B0h 14h  SHA1-HMAC of the banner for TWL/NTR titles. This can be set
              by NSS:SetTWLBannerHMAC
  4C4h ..   unknown/unspecified
  500h 40h  This is used by APT:LoadSysMenuArg and APT:StoreSysMenuArg
  540h ..   unknown/unspecified
  D70h 290h Config data struct for LGY FIRM
```

Flags from offset 0x400:

```
  +00h    1       This can be used for overriding the default FCRAM
                    memory-regions allocation sizes (APPLICATION, SYSTEM, and
                    BASE). The values for this is the same as
                    Configmem-APPMEMTYPE. Values 0-1 are handled the same way
                    by the kernel. However for NS, 0=titleinfo structure for
                    launching a title isn't set, while non-zero=titleinfo
                    structure is set.
```

```
  +01h    3       Setting bit0 here enables overriding the FIRM_* fields in
                    Configuration_Memory.
```

Config struct for booting LGY FIRMs from offset 0xD70:

```
  000h 1    Config block 0x30000
  001h 1    Config block 0x70001
  002h 1    System language (Config block 0xA0002)
  003h 1    Region from SecureInfo ("pseudo-block" 0x140000 in LGY FIRM)
  004h 0Fh  Serial number from SecureInfo ("pseudo-block" 140001h in LGY FIRM)
  013h 1    Config block 0x100002
  014h 10h  Config block 0x100003
  024h 2    Config block 0x100000
  026h 1    Cleared to zero
  027h 1    Cleared to zero
  028h 94h  Config block 0x100001
  0BCh 2    Config block 0x50000
  0BEh 2    Config block 0x50001
  0C0h 38h  Config block 0x50002
  0F8h 20h  Config block 0x50004
  118h 134h Config block 0x20000
  24Ch 10h  Config block 0x40000
  25Ch 1Ch  Config block 0x40001
  278h 4    Cleared to zero
  27Ch 4    Cleared to zero
  280h 8    Config block 0x30001
  288h 2    CRC16 over the above fields from offset 0x0, size 0x288,
              if not valid, LGY FIRM uses dummy data from .(ro)data
  28Ah 2    If non-zero, the size (below) is hardcoded (currently) to
              value 0x288, otherwise the size field below is used
  28Ch 4    Value 0x288 (size used for verifying the CRC16)
```

"Cleared to zero" fields above are not read at all by LGY FIRM.

3DS NCCH Format

NCCH (Nintendo Content Container Header) (in .app files)

This format is used to store the content of any installed title. There are two NCCH variants:

```
  CXI aka CTR Executable Image, with ARM11 code
  CFA aka CTR File Archive, with data only
```

The CXI/CFA format is structured in the following order:

```
  NCCH header                                                   (unencrypted)
  Exheader (CXI only)                                           (encrypted)
  Logo (5.0.0-11 and up)                                        (unencrypted)
  SDK Strings (aka plain binary region) (optional, CXI only)    (unencrypted)
  ExeFS filesystem (optional) - ARM11 code (CXI), icon/banner   (encrypted)
  RomFS filesystem (optional) - Used for external file storage (encrypted)
```

User-executable Tools (eg. System Settings) do have two NCCH .app files: One with the program .code, and one with the data manual. Internal System Modules (eg. MCU driver) have only one .app file (with program .code, without manual).

NCCH Header

```
  000h 100h RSA-2048 SHA-256 signature of the NCCH header
  100h 4    ID "NCCH"
  104h 4    Content size, in media units (1 media unit = 200h bytes)
  108h 8    Partition ID (usually same as Program ID, or weird stuff...?)
  110h 2    Maker code
  112h 2    Version (0001h=Weird?, 0002h=Normal)
  114h 4    SEEDDB Checksum (see NCCH Encryption chapter) (firmware 9.6.0)
  118h 8    Program ID (aka Title ID)
  120h 10h  Reserved
  130h 20h  Logo Region SHA-256 hash (5.0.0-11 and up)
```

```
  150h 10h  Product code ("CTR-x-xxxx") (for CFA: always "CTR-P-CTAP"?)
  160h 20h  Extended header SHA-256 hash (SHA256 of 2x Alignment Size... uh?)
  180h 4    Extended header size minus 400h, in bytes (usually 400h) (0=None)
  184h 4    Reserved
  188h ..   "Flags" (see below) (aka "ncchflag[0..7]")
  188h 3    Unknown (zero)
  18Bh 1    Crypto 2nd Keyslot (00h=None, 01h=Key25h, 0Ah=Key18h, 0Bh=Key1Bh)
  18Ch 1    Content Platform: 1=CTR, 2=Snake (New 3DS).
  18Dh 1    Content Type Bit-masks: Data=0x1, Executable=0x2, SystemUpdate=0x4,
              Manual=0x8, Child=(0x4|0x8), Trial=0x10.
              When 'Data' is set, but not 'Executable', NCCH is a CFA.
              Otherwise when 'Executable' is set, NCCH is a CXI.
  18Eh 1    Content Unit Size (200h SHL N)   ;uh, NOT same as "media units"?
  18Fh 1    Flags Bit0=FixedKey, Bit1=NoMountRomFs, Bit2=NoCrypto, Bit5=NewKeyY
  190h 4    Plain region offset, in media units
  194h 4    Plain region size, in media units
  198h 4    Logo Region offset, in media units ;\For applications built with
  19Ch 4    Logo Region size, in media units   ;/SDK 5+ (5.0.0-11 and up)
  1A0h 4    ExeFS offset, in media units
  1A4h 4    ExeFS total size, in media units
  1A8h 4    ExeFS header size, in media units (for SHA256 at [1C0h])
  1ACh 4    Reserved
  1B0h 4    RomFS offset, in media units
  1B4h 4    RomFS total size, in media units
  1B8h 4    RomFS header size, in media units (for SHA256 at [1E0h])
  1BCh 4    Reserved
  1C0h 20h  ExeFS superblock SHA-256 hash
  1E0h 20h  RomFS superblock SHA-256 hash
```

All of the hashes stored in this NCCH header are over the cleartext data.
Given offsets are based on the start of the file.

Extended header
The extended header contains additional information regarding/restricting access control.
3DS NCCH Extended Header
"Currently, only ExeFS:/.code can be compressed (with a LZ77 variant). A flag in the exheader determines if this is the case."
"On retail for SD applications, exheader_systeminfoflags.flag bit1 must be set."

Logo
3DS NCCH Logo

Plain binary region (SDK strings) (CXI only)
This region contains several zero-terminated ASCII strings (usually in form "[SDK+Blah:Blahblah]",00h).

```
  "The version used for the "FIRMWARE" tag (what?) is the kernel/FIRM
  version, this version can also be stored in the exheader "kernel release
  version" ARM11 kernel descriptor field. As of 2.2.0-X the NATIVE_FIRM
  kernels check the CXI exheader "kernel release version" field, if it is
  stored in the CXI exheader. If the kernel/FIRM version specified by this
  field is higher than the version of the running NATIVE_FIRM, the kernel
  will return error-code D9001413h."
```

Uh, all that blurb seems to say that... the SDK string is irrelevant - and if there's anything relevant, then it's the descriptor, but not/never the SDK string?

ExeFS and RomFS (Filesystems)
3DS NCCH ExeFS
3DS NCCH RomFS
Both ExeFS and RomFS are designed as read-only filesystems (in lack of a FAT, changing filesizes would require to relocate all following files).

Encryption (for Exheader, ExeFS, and RomFS)

[3DS NCCH Encryption]

Region Locking
The region-locking info checked by home menu is stored in the "icon" file in ExeFS.

RSA
CFAs NCCH header signature uses a fixed RSA public key.
CXIs NCCH header signature uses the RSA public key stored in the Exheader (whereof, that part of the Exheader is signed via another RSA key).

Tools
ctrtool - (CMD) (Windows/Linux) Parsing and decrypting (debug only) NCCH files

3DS NCCH Extended Header

The NCCH Exheader contains information for the .code file in the ExeFS (the .code file itself has no file header, and constains of raw code/data).

The Exheader exists only if NCCH[180h]>0. That is, it does exist for NCCH's with ExeFS .code file. For some reason ExeFS .firm does also have Exheader (but unknown if it's used for anything). Whilst E-Manuals usually have only RomFS, without ExeFS, and without Exheader.

NCCH Extended Header (Exheader, at offset 200h and up)
```
  200h 200h  SCI, System Control Info
  400h 200h  ACI, Access Control Info 1
  600h 100h  RSA-SHA256 Signature across [900h..BFFh] (using key from bootrom)
  700h 100h  RSA Public Key for NCCH Header at [000h..1FFh]
  800h 200h  ACI, Access Control Info 2 (for limitation of first ACI)
```

When loading the exheader, Process9 compares the exheader data with the data in the AccessDesc (note that not everything is compared here). When these don't match, an error is returned. The Process9 code handling this validation was updated with v6.0; the only change in this function seems to be the check for the "Ideal processor" field.

System Control Info (SCI)
```
  200h 8     Application title (default is "CtrApp") or module name (eg. "mcu")
  208h 5     Reserved
  20Dh 1     Flags (bit0=Compressed .code file, bit1=SDApplication)
  20Eh 2     Remaster version
  210h 4     Text code set info Address (usually 00100000h)             ;\
  214h 4     Text code set info Size in 1000h-byte page units           ; Code
  218h 4     Text code set info Size in bytes (excluding padding)       ;/
  21Ch 4     Stack size (in bytes?) (usually 1000h)                     ;-Stack
  220h 4     Read-only code set Address (should be 1000h-page aligned)  ;\
  224h 4     Read-only code set Size in 1000h-byte page units           ; Const
  228h 4     Read-only code set Size in bytes (excluding padding)       ;/
  22Ch 4     Reserved
  230h 4     Data code set info Address (should be 1000h-page aligned)  ;\
  234h 4     Data code set info Size in 1000h-byte page units           ; Data
  238h 4     Data code set info Size in bytes (excluding padding)       ;/
  23Ch 4     BSS size (in bytes?) (usually XXXXh)                       ;-BSS
  240h 30h*8 Dependency module list (several 8-byte Title IDs)
  3C0h 8     SystemInfo SaveData Size
  3C8h 8     SystemInfo Jump ID
  3D0h 30h   SystemInfo Reserved
```
Most of these fields are used in LOADER:LoadProcess.

Access Control Info (ACI)

```
 ARM11 Local System Capabilities:
  400h 8     Program ID (same as in NCCH Header)
  408h 4     Core version (The Title ID low of the required FIRM)
  40Ch 2     Flag1 and Flag2 (both implemented starting from 8.0.0-18).
  40Eh 1     Flag0
  40Fh 1     Priority
  410h 16*2  Resource limit descriptors ;1st byte controls max allowed CpuTime
  430h  8    Storage Info Extdata ID
  438h  8    Storage Info System savedata IDs
  440h  8    Storage Info Storage accessible unique IDs
  448h  8    Storage Info Filesystem Access Info (flags, see below)
  450h 32*8  Service Access Control
  550h 2*8   Extended service access control ;implemented with 9.3.0-X
  560h 15    Reserved
  56Fh 1     Resource limit category (0=APPLICATION, 1=SYS_APPLET,
                  2=LIB_APPLET,
                  3=OTHER (sysmodules running under the BASE memregion))
 ARM11 Kernel Capabilities:
  570h 70h   ARM11 Kernel Capability Descriptors (28x32bit)
  5E0h 10h   Reserved
 ARM9 Access Control:
  5F0h 15    ARM9 Access Control Descriptors
  5FFh 1     ARM9 Descriptor Version (must be 02h for original firmware,
               must be 02h or 03h for 9.3.0-X and up)
```

Flag0

```
  Bits  Description
  0-1   Ideal processor
  2-3   Affinity mask
  4-7   Old3DS system mode:
          0    =  Prod (64MB of usable application memory)
          1    =  Undefined (unusable)
          2    =  Dev1 (96MB of usable application memory)
          3    =  Dev2 (80MB of usable application memory)
          4    =  Dev3 (72MB of usable application memory)
          5    =  Dev4 (32MB of usable application memory)
          6-7  =  Undefined Same as Prod?
          8-15 =  unknown/unspecified
```

In the exheader data, the ideal processor field is a bit-index, while in the AccessDesc (the 2nd ACI at A00h and up) the ideal processor field is a bitmask. When the bit specified by the exheader field is not set in the AccessDesc field, an error is returned.
if((1 << exheaderval) & accessdescval == 0) return error

During a FIRM-launch when a TitleInfo structure was specified, the field at offset 400h in the FIRM-launch parameters is set to the SystemMode of the specified title, however in some cases other values are written there. With 8.0.0-18 NS will now check the output of PTMSYSM command 040A0000h, when the output is non-zero and a certain NS state field is value-zero, the following is executed otherwise this is skipped. With that check passed on 8.0.0-18, NS will then check (Flag2 & 0Fh). When that is value2, the output value (used for the FIRM-launcher parameter field mentioned above) is set to value7. Otherwise, when that value is non-zero, the output value is set to 6.

Flag1

```
  Bits  Description
  0     EnableL2Cache (Unknown what this actually does, New3DS-only presumably)
  1     cpuspeed_804MHz (Default "cpuspeed" when not set)
  2-7   Unused
```

In order for the exheader to have any of the above new bits set, the AccessDesc must have the corresponding bit set, otherwise the invalid-exheader error is returned.

Homebrew which runs under a title which has the above cpuspeed flag set, runs much faster on New3DS. It's unknown how exactly the system handles these flags.

When launching titles / perhaps other things with APT, NS uses PTMSYSM:ConfigureNew3DSCPU with data which originally came from these flags; NS does this regardless of what the running 3DS system is. However, due to a bug(?) in NS the value sent to that command is always either 00h or 03h. When calculating that value, the code only ever uses the cpuspeed field, not the cache field: code to actually load and check the value of the cache field appears to be missing.

Flag2

```
Bit     Description
0-3     New3DS system mode:
          0    = Legacy (use Old3DS system mode)
          1    = Prod (124MB of usable application memory)
          2    = Dev1 (178MB of usable application memory)
          3    = Dev2 (124MB of usable application memory)
          4-7  = Undefined Same as Prod?
          8-15 = unknown/unspecified
4-7     Unused
```

When in Legacy mode, the actual memory layout is the same as in New3DS Prod, except the available application memory as reported to the application is reduced to the Old3DS size.

The exheader value for the New3DS system mode value must be equal to the AccessDesc value, otherwise the invalid-exheader error is returned.

Storage Info Filesystem Access Info

```
Bit    Description
0      Category system application
1      Category hardware check
2      Category filesystem tool
3      Debug
4      TWL card backup
5      TWL NAND data
6      BOSS
7      sdmc:/
8      Core
9      nand:/ro/ (Read Only)
10     nand:/rw/
11     nand:/ro/ (Write Access)
12     Category system settings
13     Cardboard
14     Export/Import IVS
15     sdmc:/ (Write-only)
16     Switch cleanup (Introduced in 3.0.0?)
17     Savedata move (Introduced in 5.0.0)
18     Shop (Introduced in 5.0.0)
19     Shell (Introduced in 5.0.0)
20     Category home menu (Introduced in 6.0.0)
21     Seed DB (Introduced in 9.6.0-X FIRM, Home Menu
                has this bit set starting with 9.6.0-X)
22-55  Reserved
56     Not use RomFS
57     Use Extended Savedata Access
58-63  Reserved
```

When bit57 is set, the "Extdata ID" and "Storage Accessable Unique IDs" regions are used to store a total of 6 "Accessible Save IDs". Introduced in 6.0.0.

Service Access Control

This is the list of services which the process is allowed to access, this is registered with the services manager. Each service listed in the exheader must be listed in the AccessDesc, otherwise the invalid exheader error is

returned. The order of the services for exheader/AccessDesc doesn't matter. The AccessDesc can list services which are not in the exheader, but normally the service-access-control data for exheader/AccessDesc are exactly the same.
This list is submitted to SRVPM:RegisterProcess.

ARM11 Kernel Capability Descriptors
The kernel capability descriptors are passed to svcCreateProcess.
There are different descriptor types, determined by the number of leading ones in the binary value representation of bits 20-31. The different types are laid out as follows:

```
Bit31-20      Expl.
1110xxxxxxxx  Interrupt info
11110xxxxxxx  System call mask
                Bits 24-26: System call mask table index
                Bits 0-23: mask
1111110xxxxx  Kernel release version
                Bits 8-15: Major version
                Bits 0-7: Minor version
11111110xxxx  Handle table size
                Bits 0-18: size
111111110xxx  Kernel flags
                Bit  Description
                0    Allow debug
                1    Force debug
                2    Allow non-alphanum
                3    Shared page writing
                4    Privilege priority
                5    Allow main() args
                6    Shared device memory
                7    Runnable on sleep
                8-11 Memory type (1: application, 2: system, 3: base)
                12   Special memory
                13   Process has access to CPU core 2 (New3DS only)
11111111100x  Map address range
                Describes a memory mapping like the 111111111110
                descriptor, but an entire range rather than a single
                page is mapped. Another 11111111100x descriptor
                must follow this one to denote the (exclusive) end
                of the address range to map
111111111110  Map memory page
                Bits 0-19: page index to map (virtual address >> 12;
                the physical address is determined per-page according
                to Memory layout); Bit 20: Map read-only (otherwise
                read-write)
```

ARM9 Access Control Descriptors

```
Bit     Description
0       Mount nand:/
1       Mount nand:/ro/ (Write Access)
2       Mount twln:/
3       Mount wnand:/
4       Mount card SPI
5       Use SDIF3
6       Create seed
7       Use card SPI
8       SD application (Not checked)
9       Mount sdmc:/ (Write Access)
```

3DS NCCH ExeFS

ExeFS Header (max 10 files)

```
000h 10h*10  File headers for File 1..10        ;First at 000h
0A0h 20h     Reserved
0C0h 20h*10  SHA256 Hash for File 10..1         ;First at 1E0h
```
File hashes are stored in reverse order, i.e. the hash at offset 1E0h corresponds to the first ExeFS section. Each file header has a corresponding file hash, which is the SHA256 hash calculated over the entire file contents.

File headers
There are a maximum of 10 file headers in the ExeFS format (the maximum number of file headers is disputable, with makerom indicating a maximum of 8 sections and makecia indicating a maximum of 10 (uh, are homebrew tools?). From a non-SDK point of view, the ExeFS header format can hold no more than 10 file headers within the currently define size of 200h bytes). The file headers have the following structure:
```
00h 8     File name (ASCII, zeropadded) (all 00h for unused entries)
08h 4     File offset in bytes (0=Right after the 200h-byte ExeFS Header)
0Ch 4     File size in bytes
```
File offsets are non-inclusive of the header's size (200h bytes). Also, file headers which are not used are filled with all zeros.
The file offsets should be 200h-byte aligned. The file size is usually 4-byte aligned (unknown if that is actually required).

ExeFS Files
ExeFS or Executable Filesystem contains information related to the executable program, and is the part of the CXI format.
The ExeFS usually contains one or more of the following files:
```
.code   Contains ARM11 code, which can be optionally
          reverse-LZSS compressed via an exheader flag.
.firm   unknown/unspecified (supposedly FIRM with ARM9/ARM11 code)
logo    Contains distribution licensing Binary data
banner  Contains the banner which homemenu uses for this CXI
icon    Contains the icon which homemenu displays for this CXI
```
3DS NCCH Logo

.code (ARM11)
The .code file does simply contain plain code/data without any file header (instead, the .code file's load addresses and compression flag are located in the NCCH Exheader).
For the LZrev decompression function, see:
LZ Decompression Functions
The (decompressed) filesize should be a multiple of 1000h bytes (and the code/const/data sections should be on 1000h-byte page boundaries, as specified in the NCCH Exheader).

.code (GBA)
The .code files for GBA games do reportedly contain the GBA ROM-image, with an extra GBA Footer (with info on cartridge type and preferred LCD color scheme). For details:
3DS Config - ARM7 Registers (GBA/NDS/DSi Mode)
Unknown if GBA games have specific entries in NCCH header/exheader.

3DS NCCH RomFS

Overall RomFS file structure
```
00000h 5Ch  RomFS Header (see below)
00060h 20h  SHA256('s?) on the 1000h-byte block(s?) at xx000h
01000h ..   RomFS Directory/Files (see below)
xx000h ..   SHA256's on each 1000h-byte block at yy000h and up
yy000h ..   SHA256's on each 1000h-byte block at 1000h and up
```

RomFS Header (5Ch bytes)

```
000h 8   ID "IVFC",00h,00h,01h,00h
008h 4   Master Hash Size (eg. 20h) (aka Size of SHA256('s?) at 00060h?)
00Ch 8   Level 1 Logical Offset             (usually 0) (uh?)              ;\
014h 8   Level 1 Size of SHA256's at xx000h (eg. 60h aka 24A0h/1000h*20h)  ;
01Ch 4   Level 1 Block size (1 SHL N)       (usually 0Ch=1000h)            ;/
020h 4   Reserved (0)
024h 8   Level 2 Logical Offset             (usually 1000h) (uh?)          ;\
02Ch 8   Level 2 Size of SHA256's at yy000h (eg. 24A0h aka 124A20h/1000h*20h)
034h 4   Level 2 Block size (1 SHL N)       (usually 0Ch=1000h)            ;/
038h 4   Reserved (0)
03Ch 8   Level 3 Logical Offset             (eg. 4000h) (uh?)              ;\
044h 8   Level 3 Size of Directory/File area at 001000h  (eg. 124A20h)     ;
04Ch 4   Level 3 Block size (1 SHL N)       (usually 0Ch=1000h)            ;/
050h 4   Reserved (0)
054h 4   Header Size or so?                 (usually 5Ch)
058h 4   Optional info size (uh?)           (usually 0)
05Ch 4   unknown/padding                    (usually 0)
```

Unknow what the above "Logical Offsets" mean exactly, maybe the SHA256 tables at file offset xx0000h and yy000h are loaded first (and thus end up at RAM offsets 0000h and 1000h), then followed by the Directory/File area from file offset 1000h (being "logically" mapped to RAM offset 4000h), or so?
The various SHA256's are similar to the "Digest" tables in DSi cart header.

RomFS Directory/File area (usually/always? starting at 1000h)

```
000h 4    Header Length (usually 28h)
004h 4    Directory Hash Table Offset (usually 28h)
008h 4    Directory Hash Table Length
00Ch 4    Directory Table Offset (eg. 44h) (first=Root)
010h 4    Directory Table Length
014h 4    File Hash Table Offset
018h 4    File Hash Table Length
01Ch 4    File Table Offset (eg. 170h) (starting with Root)
020h 4    File Table Length (eg. 52Ch)
024h 4    File Data Offset (eg. 6A0h)
028h ..   Directory Hash Table
..   ..   Directory Table
..   ..   File Hash Table
..   ..   File Table
..   ..   File Data (each file aligned to 10h-byte boundary)
```

Note: The first Directory Table entry is the Root directory. Apart from that, there is no real requirement for the location of the other directory (and file) table entries. However, Table entries are commonly held to be sorted alphabetically. And, Siblings should be stored at continous table offsets (so all entries for one directory are grouped together; that can speed up sector loading).

Directory Table Entries

```
00h   4   Directory Table Offset of Parent Directory       (Self for Root)
04h   4   Directory Table Offset of next Sibling Directory (FFFFFFFFh=None)
08h   4   Directory Table Offset of first Child Directory  (FFFFFFFFh=None)
0Ch   4   File Table Offset of first File                  (FFFFFFFFh=None)
10h   4   Directory Table Offset of next Directory with same Hash
14h   4   Name Length in bytes (ie. L=NumChars*2) (0 for Root)
18h   L   Directory Name (16bit Unicode)
18h+L 0/2 Alignment padding
```

File Table Entries

```
00h   4   Directory Table Offset of Parent Directory
04h   4   File Table Offset of next Sibling File           (FFFFFFFFh=None)
08h   8   File Data Offset (64bit)
10h   8   File Data Size   (64bit)
18h   4   File Table Offset of next File with same Hash    (FFFFFFFFh=None)
1Ch   4   Name Length in bytes (ie. L=NumChars*2)
```

```
  20h   L   File Name (16bit Unicode)
  20h+L 0/2 Alignment padding
```

File/Directory Hash Tables (for quick lookup)

```
  00h   N*4 Offset to File/Directory with matching Chksum (FFFFFFFFh=None)
```

These tables can be used for fast lookup (faster than doing char-by-char string compares for all names in directory). For searching a specific name in a specific directory, compute a checksum on the Directory Offset and Name:

```
  chksum = ParentDirectoryOffset XOR 123456789
  for i=0 to NameLength/2-1
    chksum = (chksum ROR 5) XOR Name[i*2]  ;32bit rotate, then XOR lower 16bit
  offset = hashtable [(chksum MOD (hashtablesize/4))*4]
```

Then verify if Name(offset) and ParentDirectory entry(offset) do actually match, and else retry with offset=NextHashOffset(offset).

Note: The hashtablesize's should be roughly same or bigger than the total number of files/directories in the RomFS (multiplied by 4, as each entry is 32bit wide). Even then, there can be false matches, including matches in different directories (which can slowdown things when needing additional sector reads for obtaining the table entries for such directories).

RomFS can be used for...

```
  in conjunction with the ExeFS of a NCCH
  to contain the game manual accessible from the Home Menu
  to contain the DLP Child CIA   ;uh, is that "CIA" a homebrew fileformat?
  to contain game cartridge update data
```

3DS NCCH Encryption

The extended header, the ExeFS, and the RomFS are encrypted using AES-CTR.

Data Encryption (via AES-CTR)

The FirstKey is used for ExeFS Header, and for two ExeFS files ("icon" and "banner").
The SecondKey is used for all other ExeFS files (eg. ".code", or ".firm", etc.).
The SecondKey is also used for the whole RomFS (header, all files, and SHA256's).
XXX unknown which key is used for Exheader?
XXX also, New3DS firmware 9.1.0 seems to keep using the FirstKey... at least for system titles, maybe SecondKey is used only for shop titles?

Encryption FirstKeyX and FirstKeyY:

```
  FirstKeyX = as set by bootrom (in keyslot 2Ch)
  FirstKeyY = NCCH[000h..00Fh]  ;first 10h bytes of RSA signature
```

Encryption SecondKeyX:

```
  if NCCH[18Bh]=00h then SecondKeyX = KeyX from keyslot 2Ch (original firmware)
  if NCCH[18Bh]=01h then SecondKeyX = KeyX from keyslot 25h (firmware 7.0.0)
  if NCCH[18Bh]=0Ah then SecondKeyX = KeyX from keyslot 18h (firmware 9.3.0)
  if NCCH[18Bh]=0Bh then SecondKeyX = KeyX from keyslot 1Bh (firmware 9.6.0)
  if NCCH[18Bh]>00h AND Old3DS then always use keyslot 25h (or so?)
                               (ie. keyslot 18h and 1Bh are New3DS only?)
```

Encryption SecondKeyY:

```
  if NCCH[18Fh].bit5=0 then SecondKeyY = NCCH[000h..00Fh]  ;(original firmware)
  else   ;use SEEDDB...                                    ;(firmware v9.6.0)
    ;load SEEDDB (nand:/data/<ID0>/sysdata/0001000f/00000000),
    ;that file contains 32byte entries (with 8-byte Title ID, 16-byte seed,
    ;plus 8-byte reserved; whereof titleid might refer to ProgramID or so?),
    ;find the matching entry for current "titleid", then do:
    temp[00h..0Fh] = seed for current title (from SEEDDB file)          ;\
```

```
  temp[10h..17h] = NCCH[118h..11Fh]  ;Program ID                      ; test
  sha256(dst=temp, src=temp, srclen=18h)                              ;
  if temp[00h..03h] <> NCCH[114h..117h] then error                    ;/
  temp[00h..0Fh] = NCCH[000h..00Fh]  ;first 10h bytes of RSA signature ;\
  temp[10h..1Fh] = seed for current title (from SEEDDB file)          ; key
  sha256(dst=temp, src=temp, srclen=20h)                              ;
  SecondKeyY=temp[00h..0Fh]  ;use first 10h bytes of SHA256 as KeyY   ;/
```

Special Cases

```
if NCCH[18Fh].bit2=1                    ;\NoCrypto flag
  Exit  ;not encrypted                  ;/
if NCCH[118h..11Fh]=NCCH[400h..407h]    ;\
  SkipDecrypt (or NeedEncrypt)          ; Program ID in Header vs Exheader
  ;above can be checked only if Exheader ; currently/already decrypted
  ;does exist, ie. if NCCH[180h]<>0     ;/
if NCCH[18Fh].bit0=1  ;FixedCryptoKey flag        ;\
  if (ProgramID.MSW AND FFFFC010h)=00040010h      ; debug only, not retail:
    NormalKey?=FixedSystemKey (52h,7Ch,E6h,30h,..) ; instead of First/Second
  else                                          ; KeyX+KeyY
    NormalKey?=Zerofilled (00h's)               ;/
```

Encryption IV

```
if NCCH[112h]=01h  ;NCCH version (weird, is/was that version really used?)
  Force MediaUnitSize=1 (or 200h?) ;fixed? (instead of 200h SHL N bytes?)
  IV[00h..07h] = NCCH[108h..10Fh]    ;NCCH Partition ID (forwards!)
  IV[08h..0Bh] = Zero
  IV[0Ch..0Fh] = BigEndianAddr (200h=Exheader, [1A0h]=ExeFS, [1B0h]=RomFS)
if NCCH[112h]=02h  (or =00h, too?)   ;NCCH version
  IV[00h..07h] = NCCH[10Fh..108h]    ;NCCH Partition ID (backwards!)
  IV[08h]      = Type (01h=Exheader, 2=ExeFS, 3=RomFS)
  IV[09h..0Fh] = Zero (plus offset/10h within Exheader/ExeFS/RomFS area)
```

When starting elsewhere than BEGIN of Exheader/ExeFS/RomFS, add offset/10h to the IV value (whereas, mind that IV is big-endian in the above description; it may be easier to set the IV in little-endian mode though).

Blurp

Keyslot25h:
This keyslot is initialized by the 6.0 gamecard savegame keyY init function during boot, using a different portion of the final hash (this keyslot is separate from the one used for the 6.0 save crypto).
Keyslot18h:
This keyslot is initialized by arm9loader on New3DS starting with 8.1.0-0_New3DS, but only 9.3.0-X and later know how to use it with NCCH[18Bh].
Keyslot1Bh:
9.6.0-X's arm9loader changed the contents of keyslots 19h-1Fh; 9.6.0-X was the first time they were officially used, so this is not a breaking change (there is no content that would use the old versions of those keys).

3DS NCCH Logo

The logo was originally stored as file "logo" (in the encrypted ExeFS). Newer files have the logo stored in separate unencrypted data block (see NCCH header entries [130h,198h,19Ch]). Even in newer firmwares, the old location in "logo" file is still used by Safe Mode files (at least so on Old3DS). Unknown if the new logo block can also exist in absence of ExeFS.

"The Logo contains distribution licensing Binary data (types: Nintendo, Licensed, Published, or Distributed). Additionally it could have no license (type None). System applications and applets that interact with the Home Menu are typically set to type 'none' as they don't display that information."

Logo Compression

Reportedly "contains distribution licensing Binary data".
"The size of this logo is always 2000h-bytes."
The logo seems to be LZ11 compressed, see:
LZ Decompression Functions
This file is a LZ11 compressed DARC. The last 20h bytes of the decompressed data is a HMAC SHA256 hash over the actual DARC.
The HMAC key(s) used for the SHA256-HMAC is unknown.

Logo DARC Archive
3DS DARC Archive
The DARC contains three folders and several files (below is from Safe Mode System Updater):

```
anim
  NintendoLogo_D_00_SceneOutA.bclan  ;\
  NintendoLogo_D_00_SceneOutB.bclan  ; D
  NintendoLogo_D_00_SceneOutC.bclan  ;/
  NintendoLogo_U_00_SceneOutA.bclan  ;\
  NintendoLogo_U_00_SceneOutB.bclan  ; U
  NintendoLogo_U_00_SceneOutC.bclan  ;/
blyt
  NintendoLogo_D_00.bclyt            ;-D
  NintendoLogo_U_00.bclyt            ;-U
timg
  3dsLogo_00.bclim
  3dsLogo_01.bclim
  3dsLogo_02.bclim
  3dsLogo_03.bclim
  LT_00.bclim
  LT_01.bclim
  LTMask_00.bclim
  Nintendo_128x64.bclim
```

Unknown if other logo/darc packages contain other file/folder names.

Logo DARC Files
3DS CLYT Format
3DS CLAN Format
3DS CLIM Format

3DS DARC Archive

DARC files are archives. Containers of files.
This is (not so much) similiar to NARC (the DS archives).

DARC is used for logos (either the Logo Area specified in newer NCCH Headers, or the "logo" file in older NCCH ExeFS; either way, that DARC is LZ11 compressed).
DARC is used for manuals (with "Manual.bcma" being an uncompressed DARC, which contains several ".arc" files, which are themselves LZ10 compressed DARCs).

```
Uncompressed DARC     Manual.bcma file (e-manual)
LZ10 compressed DARC  .arc files (within .bcma file) (note: "LZ10" is LZSS)
LZ11 compressed DARC  NCCH Logo (in NCCH Header or NCCH ExeFS)
LZ11 compressed DARC  .arc.LZ files (eg. in 3DS Camera RomFS "lyt" folder)
```

DARC header

```
000h 4    ID 'darc'
004h 2    Endianess (FFFEh: Little)
006h 2    Header Size (001Ch)
008h 4    Version (01000000h)
00Ch 4    Total Size (excluding trailing 20h-byte SHA256-HMAC, if any)
```

```
 010h 4    Offset of File/Folder List (usually 1Ch)
 014h 4    Size of File/Folder List, PLUS File/Folder Name Area
 018h 4    Offset of File Data Area
 ..   ..   File/Folder List (0Ch bytes per file/folder)
 ..   ..   File/Folder Name Area (UTF-16)
 ..   ..   File Data Area (usually each file is 20h-byte aligned)
(..) (20h) Trailing SHA256-HMAC, if any (this is NOT 20h-byte aligned)
```

File/Folder List Entries

```
File entries (with name offset bit24=0):
 000h 4    File name offset (relative to begin of Name Area)
 004h 4    File offset (relative to begin if darc; NOT to begin of Data Area)
 008h 4    File length
Folder entries (with name offset bit24=1):
 000h 4    Folder Name offset+01000000h
 004h 4    Unknown (0=Root, 1=SubDir) (maybe folder nesting, ie. 2=SubSubDir?)
                                      (or maybe Parent index, ie. 1=Root?)
 008h 4    Folder End Index (entries up to excluding that index are childs)
```

The directory starts with two dummy enties for the root directory (with names "" and ".", and each with End.Index equal to the total number of directory entries; that number is important for knowing the end of the directory, and for computing the begin of the file name area, eg. as "[1Ch+8]*0Ch+1Ch").

Example

```
 64 61 72 63                          ;id "darc"
 FF FE 1C 00
 00 00 00 01
 28 65 00 00
 1C 00 00 00
 1C 04 00 00
 40 04 00 00
 00 00 00 01  00 00 00 00  15 00 00 00  ;00h folder ""
 02 00 00 01  00 00 00 00  15 00 00 00  ;01h folder "."
 06 00 00 01  01 00 00 00  09 00 00 00  ;02h folder "anim";\
 10 00 00 00  40 04 00 00  F8 00 00 00  ;03h              ;
 54 00 00 00  40 05 00 00  EC 00 00 00  ;04h              ;
 98 00 00 00  40 06 00 00  F8 00 00 00  ;05h              ;
 DC 00 00 00  40 07 00 00  CC 0C 00 00  ;06h              ;
 20 01 00 00  20 14 00 00  54 0C 00 00  ;07h              ;
 64 01 00 00  80 20 00 00  4C 02 00 00  ;08h              ;/
 A8 01 00 01  01 00 00 00  0C 00 00 00  ;09h folder "..." ;\
 B2 01 00 00  E0 22 00 00  A8 02 00 00  ;0Ah              ;
 E2 01 00 00  A0 25 00 00  3C 10 00 00  ;0Bh              ;/
 12 02 00 01  01 00 00 00  15 00 00 00  ;0Ch folder "..." ;\
 1C 02 00 00  00 36 00 00  28 04 00 00  ;0Dh              ;
 3E 02 00 00  80 3A 00 00  28 01 00 00  ;0Eh              ;
 60 02 00 00  00 3C 00 00  28 04 00 00  ;0Fh              ;
 82 02 00 00  80 40 00 00  28 08 00 00  ;10h              ;
 A4 02 00 00  00 49 00 00  A8 00 00 00  ;11h              ;
 C4 02 00 00  00 4A 00 00  28 08 00 00  ;12h              ;
 DC 02 00 00  80 52 00 00  28 02 00 00  ;13h              ;
 F4 02 00 00  00 55 00 00  28 10 00 00  ;14h              ;/
 00 00                                ;name "",0000h
 2E 00 00 00                          ;name ".",0000h
 61 00 6E 00 69 00 6D 00 00 00        ;name "anim",0000h
 ...
```

3DS CLYT Format

CLYT (.bclyt) is the layout format used on the 3DS. It stands for (Binary) CTR Layout, and is similar to the RLYT format used on the Wii.

Header

```
  00h 4     String  Signature (CLYT)
  04h 2     UInt16  Byte Order Mark
  06h 2     UInt16  Header Length (0014h)
  08h 4     UInt32  Revision (02020000h)
  0Ch 4     UInt32  File Size
  10h 4     UInt32  Nr Sections
```

lyt1 (Layout 1)

```
  00h 4     String  Signature (lyt1)
  04h 4     UInt32  Section Size
  08h 4     UInt32  Origin type (0=Classic, 1=Normal)
  0Ch 8     Vector2 Canvas Size
```

txl1 (Texture List 1)

```
  00h 4     String  Signature (txl1)
  04h 4     UInt32  Section Size
  08h 4     UInt32  Nr Textures = N
  0Ch N*4   UInt32[]   Texture Name Offsets (relative to the start of this
                         array)
  ..  ..               null terminated names
```

fnl1 (Font List 1)

```
  00h 4     String  Signature (fnl1)
  04h 4     UInt32  Section Size
  08h 4     UInt32  Nr Fonts = N
  0Ch N*4   UInt32[]   Font Name Offsets (relative to the start of this array)
  ..  ..               null terminated names
```

mat1 (Materials 1)

```
  00h 4       String  Signature (mat1)
  04h 4       UInt32  Section Size
  08h 4       UInt32  Nr Materials = N
  0Ch N*4     UInt32[]  Material Entry Offsets (relative to the start of this
                          section)
```

After this, the material entries follow.

```
  00h 14h   String    Material Name
  14h 4     RGBA8     Tev Color (Buffer Color?)
  18h 18h   RGBA8[6]  Tev Constant Colors
  30h 4     UInt32    Flags / bitfield
    Bit     Flag / data
    0-1     Nr texMap
    2-3     Nr texMatrix
    4-5     Nr texCoordGen
    6-8     Nr tevStage
    9       Has alphaCompare
    10      Has blendMode
    11      Use Texture Only
    12      Separate Blend Mode
    14      Has Indirect Parameter
    15-16   Nr projectionTexGenParameter
    17      Has Font Shadow Parameter
```

Texture Map Entry

```
  00h 2     UInt16  Texture Index
  02h 1     Byte    Bitfield
                Bit     Data
                0-1     Wrap S     (0=Clamp, 1=Repeat, 2=Mirror)
                2-3     Min Filter (0=Near, 1=Linear)
  02h 1     Byte    Bitfield        ;<-- uh, maybe at offset 03h?
                Bit     Data
```

```
              0-1     Wrap T     (0=Clamp, 1=Repeat, 2=Mirror)
              2-3     Mag Filter (0=Near, 1=Linear)
```

Texture Matrix Entry

```
  00h 8     Vector2 Translation
  08h 4     Single  Rotation
  0Ch 8     Vector2 Scale
```

TODO: texCoordGen, tevStage, alphaCompare, blendMode, etc...

pan1 (Pane 1)

```
  Offset  Size    Type    Description
  00h 4     String  Signature (pan1)
  04h 4     UInt32  Section Size
  08h 1     UInt8   Flags
                      Bit   Flag
                      0     Visible
                      1     InfluencedAlpha
                      2     LocationAdjust
  09h 1     UInt8   Origin
  0Ah 1     UInt8   Alpha
  0Bh 1     UInt8   Pane magnification flags
                      Bit   Flag
                      0     IgnorePartsMagnify
                      1     AdjustToPartsBounds
  0Ch 18h   String  Pane name
  24h 0Ch   Vector3 Translation
  30h 0Ch   Vector3 Rotation
  3Ch 8     Vector2 Scale
  48h 8     Vector2 Size
```

pic1 (Picture 1)

```
  Offset  Size    Type    Description
  00h 4     String  Signature (pic1)
  04h 4     UInt32  Section Size
  08h 4     RGBA8   Top left vertex color
  0Ch 4     RGBA8   Top right vertex color
  10h 4     RGBA8   Bottom left vertex color
  14h 4     RGBA8   Bottom right vertex color
  18h 2     UInt16  Material ID
  1Ah 2     UInt16  Nr texture coordinates = N
 Texture coordinates entry
  1Ch+N*20h 8     Vector2 Top left vertex texture coordinate
  24h+N*20h 8     Vector2 Top right vertex texture coordinate
  2Ch+N*20h 8     Vector2 Bottom left vertex texture coordinate
  34h+N*20h 8     Vector2 Bottom right vertex texture coordinate
```

txt1 (Text 1)

wnd1 (Window 1)

...?

bnd1 (Bounding 1)

```
  00h 4     String  Signature (bnd1)
  04h 4     UInt32  Section Size
  08h 2     UInt16? ?
  0Ah 2     UInt16? ?
  0Ch ? (0x10h up to 0x30h)     String  Bounding name
  3Ch 8     Vector2 ?
  44h 4     Vector2 ?
  ?   ?     ?       ?
```

pts1 (Parts 1)

...?

pas1 (Pane Start 1)
Starts a pane.

Offset	Size	Type	Description
00h	4	String	Signature (pas1)
04h	4	UInt32	Section Size (08h)

pae1 (Pane End 1)
Ends a pane.

00h	4	String	Signature (pae1)
04h	4	UInt32	Section Size (08h)

grp1 (Group 1)

00h	4	String	Signature (grp1)
04h	4	UInt32	Section Size (1Ch+(0..N)*10h)
08h	10h	String	Group Name
18h	4	UInt32	Nr Pane References (=N) (can be zero)
1Ch	10h*N	String[N]	Pane References

grs1 (Group Start 1)
Starts a group.

00h	4	String	Signature (grs1)
04h	4	UInt32	Section Size (08h)

gre1 (Group End 1)
Ends a group.

00h	4	String	Signature (gre1)
04h	4	UInt32	Section Size (08h)

Tools
Every File Explorer has limited support for viewing these

See also
http://florian.nouwt.com/wiki/index.php/CLYT_%28File_Format%29
https://github.com/Gericom/EveryFileExplorer/tree/master/3DS/NintendoWare/LYT1

3DS CLAN Format

This article is a stub. You can help 3DBrew by expanding it.

CLAN (.bclan) is the layout animation format used on the 3DS. It stands for (Binary) CTR Layout Animation, and is similar to the RLAN format used on the Wii.

Header

00h	4	String	Signature (CLAN)
04h	2	UInt16	Byte Order Mark
06h	2	UInt16	Header Length
08h	4	UInt32	Revision
0Ch	4	UInt32	File Size
10h	4	UInt32	Nr Sections

pat1 (Pattern 1)

00h	4	String	Signature (pat1)
04h	4	UInt32	Section Size
08h	4	UInt32	Flags?
0Ch	4	UInt32	?

```
  10h 4     Int32   ?
  14h 4     UInt32  ?
  18h 10h   String  Pattern name
```

pai1 (Pattern Instruction? 1)
```
  00h 4     String  Signature (pai1)
  04h 4     UInt32  Section Size
  08h 4     UInt32  ?
  0Ch 4     UInt32  Flags?
  10h 4     UInt32  Nr entries?
  14h 4     UInt32  Entries offset (relative to pai1)
```
Instruction? entry(ies)?
```
  00h 14h   String  Target panel name
  14h 4     UInt32  Flags?
  18h 4     UInt32  Animation entry offset (relative to entry)
  1Ch 4     UInt32  ? (if flag bit 1 is set)
```
Animation entry
```
  00h 4       String  Animation type
    Name    Description
    CLPA    CTR Layout Pane SRT (Scale/Rotate/Translate) animation.
    CLTS    CTR Layout Texture SRT animation.
    CLVI    CTR Layout Visibility animation.
    CLVC    CTR Layout Vertex Color animation.
    CLMC    CTR Layout Material Color animation.
    CLTP    CTR Layout Texture Pattern animation.
  04h 1       UInt8   Nr of animations
  05h 3       UInt8[3]        Padding
  ?   ?       ?       ?
```

See also
```
  http://wiki.tockdom.com/wiki/BRLAN
  http://wiibrew.org/wiki/Wii_Animations#Animations_.28.2A.brlan.29
```

3DS CLIM Format

Footer
```
 000h 4   ID "CLIM"
 004h 2   FFh,FEh
 006h 2   Header Size? (14h)
 008h 4   Version? (02020000h)
 00Ch 4   Total File size
 010h 4?  Maybe number of images? (00000001h)
 014h 4   Chunk ID "imag"       ;\
 018h 4   Chunk Size (10h)      ; image data is at offset 0
 01Ch 2   Width?                ; (if multiple images should be allowed...
 01Eh 2   Height?               ; next image is at WHICH offset?)
 020h 4?  Type? (0Dh = 4bpp?)   ;/   ;<-- ETC1A4 ?
 024h 4   Footer Offset (from begin of file to "CLIM")  ;<-- at end of file
```

3DS 3DSX Format (homebrew executables)

The 3DSX format is a homebrew executable file format. The main feature is having relocateable code, which may have been required for some older exploits that didn't provide full control of the virtual memory mapping. As such, the .3dsx format is no longer useful, and one could instead use regular .code (or .firm) files.

3DSX File Format

Header:

```
  00h  4   ID "3DSX"
  04h  2   Header size (20h or 2Ch)
  06h  2   Relocation Header size (unknown, maybe 3*(4+4) ?)
  08h  4   Format version (unknown/unspecified)
  0Ch  4   Flags          (unknown/unspecified)
  10h  4   Code Segment size
  14h  4   Rodata segment size
  18h  4   Data segment size (including bss)
  1Ch  4   Bss segment size
```

Extended Header (when [04h]>20h):

```
 (20h) 4   Icon/Title Offset (SMDH Format)
 (24h) 4   Icon/Title Size (36C0h)
 (28h) 4   RomFS Offset
```

Relocation Header(s)

```
  ..   4   Number of absolute Code relocations
  ..   4   Number of relative Code relocations
  ..   4   Number of absolute Rodata relocations
  ..   4   Number of relative Rodata relocations
  ..   4   Number of absolute Data relocations
  ..   4   Number of relative Data relocations
```

Segments

```
  ..   ..  Code Segment
  ..   ..  Rodata Segment
  ..   ..  Data Segment
```

Relocation Tables

```
  ..   ..  Code Relocation Table
  ..   ..  Rodata Relocation Table
  ..   ..  Data Relocation Table
```

Optional (when [04h]>20h):

```
  ..   ..  Icon/Title (SMDH Format)
  ..   ..  RomFS Offset
```

Relocation Table (entries?):

```
  00h 2   Number of words to skip
  02h 2   Number of words to patch
```

Unknown what this does. The file is probably meant to be originated at a fixed base address (maybe zero?), and one would add the actual load address to any such "absolute" addresses? And do whatever on "relative" addresses?

3DS CCAL Format (Hardware calibration, HWCAL)

Reading

"If 0x1FF81006 (uh, what, where from?) is 3 or 4 or 7 or 8 or 9 then the calibration data is read from the EEPROM using the i2c:EEP service command 0x001000C0, using offset 0x000 for HWCAL0, and offset 0x800 for HWCAL1. Otherwise attempt is made to read CTRNAND:/ro/sys/HWCAL(0|1).dat instead."

Summary

File	Data	Size	ConfigInfoBlk	Description
000h	-	200h	-	Header
200h	000h	0x10	00040000h	Touchscreen
214h	014h	0x08	???	Circle Pad (part 1)
220h	020h	2*	00050000h	Screen flicker
224h	024h	1*	???	RTC
228h	028h	1*	???	DSPRAM related
230h	030h	0x8A	???	Camera position
2BCh	0BCh	0x12	???	Gyroscope
2D0h	0D0h	0xC	???	Accelerometer

```
2E0h 0E0h  0x134   ???            Codec (CDC)
418h 218h  0x06    ???            Programmable Infrared Transmitter (PIT)
420h 220h  0x214   ???            3D filters
640h 440h  0x20    ???            Power saving mode (ABL)
670h 470h  0x20    ???            ???
6A0h 4A0h  0x38    00050002h      Backlight PWM (BLPWM)
6E0h 4E0h  0x18    ???            Circle Pad extra (part 2)
700h 500h  0xC     ???            ???
710h 510h  0x20    ???            ???
740h 540h  0x08    ???            MCU
750h 550h  0x02    ???            3D screen (ULCD) delay
760h 560h  0x08    ???            Microphone echo cancellation
770h 570h  0x10C   ???            Power saving mode (ABL) extra
880h 680h  0x08    ???            CStick (Right stick)
890h 690h  0x18    ???            Quad Tracking Module (QTM)
```

Header

```
000h 4    File ID "CCAL"
004h 4    Version (eg. Old3DS:0000000Dh, New3DS:00000012h)
008h 4    Data size (always 07D0h, for entry 200h..9CFh)
00Ch 1    Model version (?) (usually 00h)
00Dh 1    CAL revision (incremented each time the CAL file is updated)
00Eh 2    Bitmask of successful Aging tests (Old3DS:73FEh, New3DS:77FEh)
010h 20h  SHA256HMAC (retail) or SHA256 (dev) of the data section
030h 1D0h Zero
```

Touchscreen

```
200h 2    s16 RawX0
202h 2    s16 RawY0
204h 2    s16 PointX0
206h 2    s16 PointY0
208h 2    s16 RawX1
20Ah 2    s16 RawY1
20Ch 2    s16 PointX1
20Eh 2    s16 PointY1
210h 2    Checksum?
212h 2    Zero
```

Circle Pad (part 1)

```
214h 2    s16 CenterX (usually 08xxh)
216h 2    s16 CenterY (usually 08xxh)
218h 4    Zero
21Ch 2    Checksum?
21Eh 2    Zero
```

Screen flicker

```
220h 1    u8 FlickerTop                          (maybe for MCU[03h] ?)
221h 1    u8 FlickerBottom                       (maybe for MCU[04h] ?)
222h 1    Checksum ([220h] XOR FFh)
223h 1    Checksum ([221h] XOR FFh)    ;uh, reportedly without XOR FFh ???
```

RTC

```
224h 1    u8 CompensationValue (???) (usually 80h) (maybe for MCU[37h] ?)
225h 1    Checksum ([224h] XOR FFh)
226h 2    Zero
```

DSP-RAM related?

```
228h 1    DSPRAM related (spotted FEh on Old3DS, 86h on New3DS)
229h 1    Checksum ([228h] XOR FFh)
22Ah 6    Zero
```

Camera position ;uh, for which of the three camera(s)?

```
230h 4    u32 flags
234h 4    float scale
238h 4    float RotationZ
23Ch 4    float TranslationX
240h 4    float TranslationY
244h 4    float RotationX
248h 4    float RotationY
24Ch 4    float ViewAngleRight
250h 4    float ViewAngleLeft
254h 4    float ChartDistance(???)
258h 4    float CameraDistance
25Ch 2    s16 ImageWidth
25Eh 2    s16 ImageHeight
260h 10h  u8 reserved[0x10]
270h 40h  u8 ???[0x40]
2B0h 2    s16 aeBaseTarget(???)
2B2h 2    s16 kRL
2B4h 2    s16 kGL
2B6h 2    s16 kBL
2B8h 2    s16 ccmPosition
2BAh 2    Checksum?
```

Gyroscope

```
2BCh 2    s16 ZeroX
2BEh 2    s16 PlusX
2C0h 2    s16 MinusX
2C2h 2    s16 ZeroY
2C4h 2    s16 PlusY
2C6h 2    s16 MinusY
2C8h 2    s16 ZeroZ
2CAh 2    s16 PlusZ
2CCh 2    s16 MinusZ
2CEh 2    Checksum?
```

Accelerometer

```
2D0h 2    s16 OffsetX
2D2h 2    s16 ScaleX
2D4h 2    s16 OffsetY
2D6h 2    s16 ScaleY
2D8h 2    s16 OffsetZ
2DAh 2    s16 ScaleZ
2DCh 2    Checksum?
2DEh 2    Zero
```

Codec (CDC) (aka TSC)

```
2E0h 1    u8 DriverGainHP       ;TSC[65h:0Ch].bit3-7
2E1h 1    u8 DriverGainSP       ;TSC[65h:12h].bit2-7 and TSC[65h:13h].bit2-7
2E2h 1    u8 AnalogVolumeHP     ;TSC[65h:16h] and TSC[65h:17h]
2E3h 1    u8 AnalogVolumeSP     ;TSC[65h:1Bh] and TSC[65h:1Ch]
2E4h 1    s8 ShutterVolume0     ;TSC[00h:41h] and TSC[00h:42h] ;\maybe camera
2E5h 1    s8 ShutterVolume1     ;TSC[64h:7Bh]                  ;/sound fx?
2E6h 1    u8 MicrophoneBias     ;TSC[65h:33h]
2E7h 1    u8 QuickCharge (???)  ;TSC[65h:42h].bit0-1
2E8h 1    u8 PGA_GAIN (mic)     ;TSC[65h:41h].bit0-5
2E9h 3    u8 reserved[3]
2ECh 1Eh  s16 FilterHP32[3*5]   ;TSC[0Bh:02h..1Fh] and TSC[0Bh:42h..5Fh]
30Ah 1Eh  s16 FilterHP47[3*5]   ;TSC[0Bh:20h..3Dh] and TSC[0Bh:60h..7Dh]
328h 1Eh  s16 FilterSP32[3*5]   ;TSC[0Ch:02h..1Fh] and TSC[0Ch:42h..5Fh]
346h 1Eh  s16 FilterSP47[3*5]   ;TSC[0Ch:20h..3Dh] and TSC[0Ch:60h..7Dh]
364h 38h  s16 FilterMic32[3+5*5];TSC[05h:08h..3Fh]
39Ch 38h  s16 FilterMic47[3+5*5];TSC[05h:48h..7Fh]
3D4h 38h  s16 FilterFree [3+5*5];TSC[08h:xxh, 09h;xxh, 0Ah:xxh]
40Ch 1    u8 AnalogInterval     ;TSC[67h:27h].lsbs ;\
40Dh 1    u8 AnalogStabilize    ;TSC[67h:19h].lsbs ;
```

```
  40Eh 1    u8 AnalogPrecharge    ;TSC[67h:17h].msbs ; Touchscreen/CirclePad
  40Fh 1    u8 AnalogSense        ;TSC[67h:17h].lsbs ;
  410h 1    u8 AnalogDebounce     ;TSC[67h:1Bh].lsbs ;
  411h 1    u8 Analog_XP_Pullup   ;TSC[67h:19h].msbs ;
  412h 1    u8 YM_Driver          ;TSC[67h:1Bh].msb  ;/
  413h 1    u8 reserved
  414h 2    Checksum?
  416h 2    Zero
```

Programmable Infrared Transmitter (PIT) -- New3DS only (zerofilled on Old3DS)

```
  418h 2    u16 VisibleFactor
  41Ah 2    u16 IRFactor
  41Ch 2    Zero
  41Eh 2    Checksum?
```

3D filters ;uh, is that 3d video? and/or 3d surround audio?

```
  420h 200h u16 SpecialFilter[0x100] <-- Old3DS: mostly zero, unlike New3DS
  620h 14h  u32 IIRSurroundFilter[5] <-- both Old3DS and New3DS
  634h 2    Checksum?
  636h 0Ah  Zero
```

Power saving mode (ABL)

```
  640h 4    u32 DitherPattern
  644h 2    s16 StartX
  646h 2    s16 StartY
  648h 2    u16 SizeX
  64Ah 2    u16 SizeY
  64Ch 2    s16 GTH_Ratio
  64Eh 1    u8 DitherMode
  64Fh 1    u8 MinRS
  650h 1    u8 MaxRS
  651h 1    u8 MinGTH
  652h 1    u8 MinMax (???)
  653h 1    u8 ExMax (???)
  654h 1    u8 inertia
  655h 9    u8 LutListRS[9]
  65Eh 2    u8 reserved[2]
  660h 2    Checksum?
  662h 0Eh  Zero
```

Unknown ???

```
  670h 20h  ???
  690h 2    Checksum?
  692h 0Eh  Zero
```

Backlight PWM (BLPWM) ;uh, which backlight/mode(s)? Upper2D, Upper3D, Lower?

```
  6A0h 24h  float coefficient[3][3]
  6C4h 1    u8 NumLevels
  6C5h 1    u8 padding
  6C6h 0Eh  u16 brightnesses[7]
  6D4h 2    u16 BaseDivisor
  6D6h 2    u16 MinimumBrightnessHw
  6D8h 2    Checksum?
  6DAh 06h  Zero
```

Circle Pad (part 2)

```
  6E0h 4    float ScaleX
  6E4h 4    float ScaleY
  6E8h 2    s16 MaxX
  6EAh 2    s16 MinX
  6ECh 2    s16 MaxY
  6EEh 2    s16 MinY
```

```
  6F0h 2    s16 type
  6F2h 6    u8 unknown_padding[6]
  6F8h 2    Checksum?
  6FAh 06h  Zero
```

Unknown ???

```
  700h 0Ch  ???
  70Ch 2    Checksum?
  70Eh 2    Zero
```

Unknown ???

```
  710h 20h  ???
  730h 2    Checksum?
  732h 0Eh  Zero
```

MCU

```
  740h 2    s16 SVR2_Min (???)
  742h 2    s16 SVR2_Max (???)
  744h 2    s16 VolumeSliderMin      (maybe for MCU[58h] ?)
  746h 2    s16 VolumeSliderMax      (maybe for MCU[59h] ?)
  748h 2    Value 0001h (?)
  74Ah 2    Checksum?
  74Ch 4    Zero
```

3D screen (ULCD) delay

```
  750h 1    u8 To2D
  751h 1    u8 To3D
  752h 2    Value 0001h (?)
  754h 2    Checksum?
  756h 0Ah  Zero
```

Microphone echo cancel

```
  760h 8    s8 params[8]
  768h 2    Value 0001h (?)
  76Ah 2    Checksum?
  76Ch 4    Zero
```

Power saving mode (ABL) extra -- New3DS only (zerofilled on Old3DS)

```
  770h 1    u8 MaxInertia
  771h 1    u8 pad
  772h 2    u16 PWM_CNT_EX
  774h 4    u32 Histogram1
  778h 4    u32 Histogram2
  77Ch 100h u32 adjust[0x40]
  87Ch 2    Value 0001h (?)
  87Eh 2    Checksum?
```

CStick (Right stick) -- New3DS only (zerofilled on Old3DS)

```
  880h 1    u8 ThinningCountX(???)
  881h 1    u8 ThinningCountY(???)
  882h 6    ` u16 reserved[3]
  888h 2    Value 0001h (?)
  88Ah 2    Checksum?
  88Ch 4    Zero
```

Quad Tracking Module (QTM) -- New3DS only (zerofilled on Old3DS)

```
  890h 4    float DivisorAtZero (???)
  894h 4    float TranslationX
  898h 4    float TranslationY
  89Ch 4    float RotationZ
  8A0h 4    float HorizontalAngle
  8A4h 4    float OptimalDistance
```

```
8A8h ..   ?
8A8h 2    Zero
8AAh 2    Checksum?
8ACh 124h Zero
```

Unknown things
The HMAC key for header signature is unknown.
The Checksum function is unknown (for 1-2 byte sections it's just the 1-2 byte(s) XORed FFh).
The purpose of most entries is unknown.
Unknown how to read the EEPROM data (maybe similar as for Wifi EEPROM?).
Unknown what size EEPRM has (apparently at least 1000h bytes).
Unknown if EEPROM has any header (maybe only 30h bytes instead 200h bytes?).

Aging flags (for entry [00Eh])

Bit	Description
0	FCRAM
1	LCD flicker (always successful)
2	Camera
3	Touch panel (always successful)
4	Circle Pad (analog stick)
5	Codec
6	Gyroscope
7	RTC
8	Accelerometer
9	Surround
10	Power saving mode (ABL)
11	3D screen (ULCD)
12	Backlight PWM
13	Analog stick A (???)
14	Camera extensions
15	Power saving mode (ABL) in legacy (DSi/GBA) mode

Unknown if these are actually test results, maybe they are just entry-present flags? With later entries instead indicated by "Value 0001h" in the newly added fields?

3DS Title IDs

3DS Partition Files/Folders

```
3ds:\__journal.nn_                      file?
3ds:\data\<ID0>\extdata\00048000\..     savedata files  ;\note: <ID0> comes
3ds:\data\<ID0>\sysdata\..              savedata files  ;/from movable.sed
3ds:\dbs\certs.db                       database?
3ds:\dbs\import.db                      database?
3ds:\dbs\ticket.db                      database? (insane 37Mbyte file???)
3ds:\dbs\title.db                       database?
3ds:\dbs\tmp_i.db                       database?
3ds:\dbs\tmp_t.db                       database?
3ds:\fixdata\sysdata\                   empty folder
3ds:\private\movable.sed                file?
3ds:\ro\private\                        empty folder
3ds:\ro\shared\                         empty folder
3ds:\ro\sys\HWCAL0.dat                  hardware calibration data
3ds:\ro\sys\HWCAL1.dat                  hardware calibration data copy
3ds:\rw\shared\                         empty folder
3ds:\rw\sys\lgy.log                     log file for DSi ErrDisp?
3ds:\rw\sys\LocalFriendCodeSeed_B       seed file?
3ds:\rw\sys\native.log (if any)         log file for 3DS ErrDisp?
3ds:\rw\sys\rand_seed                   seed file?
3ds:\rw\sys\SecureInfo_A                Region and Serial/Barcode
3ds:\rw\sys\updater.log                 log file?
3ds:\ticket\                            empty folder
```

```
3ds:\title\00040010\..                  System Applications
3ds:\title\0004001b\..                  System Data Archives 1
3ds:\title\00040030\..                  System Applets
3ds:\title\0004009b\..                  Shared Data Archives
3ds:\title\000400db\..                  System Data Archives 2
3ds:\title\00040130\..                  System Modules
3ds:\title\00040138\..                  System Firmware
3ds:\tmp\title.tik                      temporary file?
```

DSi 1st Partition Files/Folders on 3DS

```
dsi:\import\                            empty folder
dsi:\shared1\                           empty folder
dsi:\shared2\0000                       sound recorder data
dsi:\sys\log\inspect.log                log file
dsi:\sys\log\inspect.log~               log file
dsi:\sys\log\product.log                log file
dsi:\sys\log\product.log~               log file
dsi:\ticket\                            empty folder
dsi:\title\00030005\42383841\..         DS Internet settings for 3DS
dsi:\title\00030005\484e4441\..         DS Download Play
dsi:\title\0003000f\484e4841\..         DS Cart Whitelist
dsi:\title\0003000f\484e4c41\..         DSi Version Data
dsi:\tmp\                               empty folder
```

DSi 2nd Partition Files/Folders on 3DS

```
photo:\photo\private\ds\app\484E494A\pit.bin   camera info
```

3ds:\title\00040010\.. System Applications

```
00020000 System Settings, mset JPN      ;\
00021000 System Settings, mset USA      ;
00022000 System Settings, mset EUR      ; System Settings (mset)
00026000 System Settings, mset CHN      ; CTR-N-HASx
00027000 System Settings, mset KOR      ;
00028000 System Settings, mset TWN      ;/
00020100 Download Play, dlplay JPN      ;\
00021100 Download Play, dlplay USA      ;
00022100 Download Play, dlplay EUR      ; Download Play (dlplay)
00026100 Download Play, dlplay CHN      ; CTR-N-HDLx
00027100 Download Play, dlplay KOR      ;
00028100 Download Play, dlplay TWN      ;/
00020200 Activity Log JPN               ;\
00021200 Activity Log USA               ;
00022200 Activity Log EUR               ; Activity Log
00026200 Activity Log CHN               ; CTR-N-HMKx
00027200 Activity Log KOR               ;
00028200 Activity Log TWN               ;/
00020300 Health Safety, safe JPN        ;\
00021300 Health Safety, safe USA        ;
00022300 Health Safety, safe EUR        ; Health and Safety Info (safe)
00026300 Health Safety, safe CHN        ; CTR-N-HACx
00027300 Health Safety, safe KOR        ;
00028300 Health Safety, safe TWN        ;
20020300 Health Safety, safe JPN New3DS ;
20021300 Health Safety, safe USA New3DS ;
20022300 Health Safety, safe EUR New3DS ;
N/A      Health Safety, safe CHN New3DS ;
20027300 Health Safety, safe KOR New3DS ;
N/A      Health Safety, safe TWN New3DS ;/
00020400 Nintendo 3DS Camera, CtrApp JPN ;\
00021400 Nintendo 3DS Camera, CtrApp USA ;
00022400 Nintendo 3DS Camera, CtrApp EUR ; Nintendo 3DS Camera (CtrApp)
00026400 Nintendo 3DS Camera, CtrApp CHN ; CTR-N-HEPx
00027400 Nintendo 3DS Camera, CtrApp KOR ;
00028400 Nintendo 3DS Camera, CtrApp TWN ;/
```

```
00020500 Nintendo 3DS Sound, CtrApp JPN  ;\
00021500 Nintendo 3DS Sound, CtrApp USA  ;
00022500 Nintendo 3DS Sound, CtrApp EUR  ; Nintendo 3DS Sound (CtrApp)
00026500 Nintendo 3DS Sound, CtrApp CHN  ; CTR-N-HESx
00027500 Nintendo 3DS Sound, CtrApp KOR  ;
00028500 Nintendo 3DS Sound, CtrApp TWN  ;/
00020700 Mii Maker, EDIT JPN             ;\
00021700 Mii Maker, EDIT USA             ;
00022700 Mii Maker, EDIT EUR             ; Mii Maker (EDIT)
00026700 Mii Maker, EDIT CHN             ; CTR-N-HEDx
00027700 Mii Maker, EDIT KOR             ;
00028700 Mii Maker, EDIT TWN             ;/
00020800 StreetPass Mii Plaza, MEET JPN  ;\
00021800 StreetPass Mii Plaza, MEET USA  ;
00022800 StreetPass Mii Plaza, MEET EUR  ; StreetPass Mii Plaza (MEET)
00026800 StreetPass Mii Plaza, MEET CHN  ; CTR-N-HMEx
00027800 StreetPass Mii Plaza, MEET KOR  ;
00028800 StreetPass Mii Plaza, MEET TWN  ;/
00020900 eShop, tiger JPN                ;\
00021900 eShop, tiger USA                ;
00022900 eShop, tiger EUR                ; eShop (tiger)
N/A      eShop, tiger CHN                ; CTR-N-HGRx
00027900 eShop, tiger KOR                ;
00028900 eShop, tiger TWN                ;/
00020a00 System Transfer, CARDBOARD JPN  ;\
00021a00 System Transfer, CARDBOARD USA  ;
00022a00 System Transfer, CARDBOARD EUR  ; System Transfer (CARDBOARD)
N/A      System Transfer, CARDBOARD CHN  ; CTR-N-HCBx
00027a00 System Transfer, CARDBOARD KOR  ;
00028a00 System Transfer, CARDBOARD TWN  ;/
00020b00 Nintendo Zone, Nintendo JPN     ;\
00021b00 Nintendo Zone, Nintendo USA     ;
00022b00 Nintendo Zone, Nintendo EUR     ; Nintendo Zone ("Nintendo")
N/A      Nintendo Zone, Nintendo CHN     ; CTR-N-HMAx
N/A      Nintendo Zone, Nintendo KOR     ;
N/A      Nintendo Zone, Nintendo TWN     ;/
00020d00 Face Raiders JPN                ;\
00021d00 Face Raiders USA                ;
00022d00 Face Raiders EUR                ; Face Raiders
00026d00 Face Raiders CHN                ; CTR-N-HCHx
00027d00 Face Raiders KOR                ;
00028d00 Face Raiders TWN                ;
20020d00 Face Raiders JPN New3DS         ;
20021d00 Face Raiders USA New3DS         ;
20022d00 Face Raiders EUR New3DS         ;
N/A      Face Raiders CHN New3DS         ;
20027d00 Face Raiders KOR New3DS         ;
N/A      Face Raiders TWN New3DS         ;/
00020e00 AR Games, AR_ACT JPN            ;\
00021e00 AR Games, AR_ACT USA            ;
00022e00 AR Games, AR_ACT EUR            ; AR Games (AR_ACT)
00026e00 AR Games, AR_ACT CHN            ; CTR-N-HARx
00027e00 AR Games, AR_ACT KOR            ;
00028e00 AR Games, AR_ACT TWN            ;/
00020f00 Safe mode SysUpdater, mset JPN  ;\
00021f00 Safe mode SysUpdater, mset USA  ;
00022f00 Safe mode SysUpdater, mset EUR  ; Safe mode System Updater (mset)
00026f00 Safe mode SysUpdater, mset CHN  ; CTR-N-HSHx
00027f00 Safe mode SysUpdater, mset KOR  ;
00028f00 Safe mode SysUpdater, mset TWN  ;/
00023000 Promotional video JPN           ;\
00024000 Promotional video USA           ;
00025000 Promotional video EUR           ; Promotional video
N/A      Promotional video CHN           ; (Variable?)
N/A      Promotional video KOR           ;
```

```
N/A      Promotional video TWN           ;/
0002bf00 Network ID Settings, act JPN    ;\
0002c000 Network ID Settings, act USA    ;
0002c100 Network ID Settings, act EUR    ; Nintendo Network ID Settings (act)
N/A      Network ID Settings, act CHN    ; CTR-N-HAFx
N/A      Network ID Settings, act KOR    ;
N/A      Network ID Settings, act TWN    ;/
20023100 New3DS microSD Management, mcopy JPN  ;\
20024100 New3DS microSD Management, mcopy USA  ; New_3DS-only
20025100 New3DS microSD Management, mcopy EUR  ; microSD Management ('mcopy')
N/A      New3DS microSD Management, mcopy CHN  ; CTR-N-HAJx
N/A      New3DS microSD Management, mcopy KOR  ;
N/A      New3DS microSD Management, mcopy TWN  ;/
2002c800 New3DS HOME menu/menu JPN       ;\
2002cf00 New3DS HOME menu/menu USA       ; New_3DS-only, currently stubbed
2002d000 New3DS HOME menu/menu EUR       ; "HOME menu/menu"
N/A      New3DS HOME menu/menu CHN       ; Contains information manual data
2002d700 New3DS HOME menu/menu KOR       ;
N/A      New3DS HOME menu/menu TWN       ;/ CTR-P-CTAP
2002c900 New3DS Friends list/friend JPN  ;\
2002d100 New3DS Friends list/friend USA  ; New_3DS-only, currently stubbed
2002d200 New3DS Friends list/friend EUR  ; "Friends list/friend"
N/A      New3DS Friends list/friend CHN  ; Contains information manual data
2002d800 New3DS Friends list/friend KOR  ;
N/A      New3DS Friends list/friend TWN  ;/ CTR-P-CTAP
2002ca00 New3DS Notifications/newslist JPN ;\
2002d300 New3DS Notifications/newslist USA ; New_3DS-only, currently stubbed
2002d400 New3DS Notifications/newslist EUR ; "Notifications/newslist"
N/A      New3DS Notifications/newslist CHN ; Contains information manual data
2002d900 New3DS Notifications/newslist KOR ;
N/A      New3DS Notifications/newslist TWN ;/ CTR-P-CTAP
2002cb00 New3DS Game notes/cherry JPN    ;\
2002d500 New3DS Game notes/cherry USA    ; New_3DS-only, currently stubbed
2002d600 New3DS Game notes/cherry EUR    ; "Game notes/cherry"
N/A      New3DS Game notes/cherry CHN    ; Contains information manual data
2002da00 New3DS Game notes/cherry KOR    ;
N/A      New3DS Game notes/cherry TWN    ;/ CTR-P-CTAP
```

3ds:\title\0004001b System Data Archives 1

```
00010002 ClCertA
00010702 NS CFA
00010802 old CFA dummy
00018002 new CFA dummy
00018102 web-browser data
00018202 web-kit/OSS CROs
00019002 Fangate_updater
```

3ds:\title\00040030 System Applets

```
00008102 Test Menu, Demo1 ALL                ;-CTR-P-CTAP
00008202 Home Menu, menu JPN                 ;\
00008f02 Home Menu, menu USA                 ;
00009802 Home Menu, menu EUR                 ; Home Menu (menu)
0000a102 Home Menu, menu CHN                 ; CTR-P-HMMx
0000a902 Home Menu, menu KOR                 ;
0000b102 Home Menu, menu TWN                 ;/
00008402 Camera applet JPN                   ;\
00009002 Camera applet USA                   ; Camera applet (CtrApp)
00009902 Camera applet EUR                   ; used by Home-menu
0000a202 Camera applet CHN                   ; CTR-N-HCSx
0000aa02 Camera applet KOR                   ;
0000b202 Camera applet TWN                   ;/
00008502  JPN                                ;\
00009102  USA                                ;
00009a02  EUR                                ; whatever
```

```
?         CHN                                ; not available on CDN
?         KOR                                ;
?         TWN                                ;/
00008602 Instruction Manual JPN              ;\
00009202 Instruction Manual USA              ;
00009b02 Instruction Manual EUR              ; Instruction Manual viewer
0000a402 Instruction Manual CHN              ; CTR-N-HMVx
0000ac02 Instruction Manual KOR              ;
0000b402 Instruction Manual TWN              ;/
00008702 Game Notes, Cherry JPN              ;\
00009302 Game Notes, Cherry USA              ;
00009c02 Game Notes, Cherry EUR              ; Game Notes (Cherry)
0000a502 Game Notes, Cherry CHN              ; CTR-N-HGMx
0000ad02 Game Notes, Cherry KOR              ;
0000b502 Game Notes, Cherry TWN              ;/
00008802 Internet Browser, spider JPN        ;\
00009402 Internet Browser, spider USA        ;
00009d02 Internet Browser, spider EUR        ; Internet Browser (spider)
0000a602 Internet Browser, spider CHN        ;
0000ae02 Internet Browser, spider KOR        ;
0000b602 Internet Browser, spider TWN        ;/
20008802 Internet Browser, SKATER JPN New3DS ;\
20009402 Internet Browser, SKATER USA New3DS ; New 3DS
20009d02 Internet Browser, SKATER EUR New3DS ; Internet Browser (SKATER)
?        Internet Browser, SKATER CHN New3DS ; CTR-N-HBRx
2000ae02 Internet Browser, SKATER KOR New3DS ;
N/A      Internet Browser, SKATER TWN New3DS ;/
00008a02 ErrDisp ALL                         ;\Fatal error viewer, ErrDisp
00008a03 ErrDisp ALL Safe mode               ;/
20008a03 ErrDisp JPN Safe mode New3DS        ;\
20008a03 ErrDisp USA Safe mode New3DS        ;
20008a03 ErrDisp EUR Safe mode New3DS        ; ErrDisp Safe mode New3DS
?        ErrDisp CHN Safe mode New3DS        ;
20008a03 ErrDisp KOR Safe mode New3DS        ;
N/A      ErrDisp TWN Safe mode New3DS        ;/
00008d02 Friend List, friend JPN             ;\
00009602 Friend List, friend USA             ;
00009f02 Friend List, friend EUR             ; Friend List (friend)
0000a702 Friend List, friend CHN             ; CTR-N-HFRx
0000af02 Friend List, friend KOR             ;
0000b702 Friend List, friend TWN             ;/
00008e02 Notifications, newslist JPN         ;\
00009702 Notifications, newslist USA         ;
0000a002 Notifications, newslist EUR         ; Notifications (newslist)
0000a802 Notifications, newslist CHN         ; CTR-N-HCRx
0000b002 Notifications, newslist KOR         ;
0000b802 Notifications, newslist TWN         ;/
0000c002 Keyboard, swkbd JPN                 ;\
0000c802 Keyboard, swkbd USA                 ;
0000d002 Keyboard, swkbd EUR                 ; Software Keyboard (swkbd)
0000d802 Keyboard, swkbd CHN                 ; CTR-N-HKYx
0000de02 Keyboard, swkbd KOR                 ;
0000e402 Keyboard, swkbd TWN                 ;
0000c003 Keyboard, swkbd JPN Safe mode       ;
0000c803 Keyboard, swkbd USA Safe mode       ;
0000d003 Keyboard, swkbd EUR Safe mode       ;
0000d803 Keyboard, swkbd CHN Safe mode       ;
0000de03 Keyboard, swkbd KOR Safe mode       ;
0000e403 Keyboard, swkbd TWN Safe mode       ;
2000c003 Keyboard, swkbd JPN Safe mode New3DS ;
2000c803 Keyboard, swkbd USA Safe mode New3DS ;
2000d003 Keyboard, swkbd EUR Safe mode New3DS ;
?        Keyboard, swkbd CHN Safe mode New3DS ;
2000de03 Keyboard, swkbd KOR Safe mode New3DS ;
N/A      Keyboard, swkbd TWN Safe mode New3DS ;/
```

```
0000c102 Mii picker, appletEd JPN            ;\
0000c902 Mii picker, appletEd USA            ;
0000d102 Mii picker, appletEd EUR            ; Mii picker (appletEd)
0000d902 Mii picker, appletEd CHN            ; CTR-N-HMSx
0000df02 Mii picker, appletEd KOR            ;
0000e502 Mii picker, appletEd TWN            ;/
0000c302 Picture picker, PNOTE_AP JPN        ;\
0000cb02 Picture picker, PNOTE_AP USA        ;
0000d302 Picture picker, PNOTE_AP EUR        ; Picture picker (PNOTE_AP)
0000db02 Picture picker, PNOTE_AP CHN        ; CTR-N-HCCx
0000e102 Picture picker, PNOTE_AP KOR        ;
0000e702 Picture picker, PNOTE_AP TWN        ;/
0000c402 Voice memo picker, SNOTE_AP JPN     ;\
0000cc02 Voice memo picker, SNOTE_AP USA     ;
0000d402 Voice memo picker, SNOTE_AP EUR     ; Voice memo picker (SNOTE_AP)
0000dc02 Voice memo picker, SNOTE_AP CHN     ; CTR-N-HMCx
0000e202 Voice memo picker, SNOTE_AP KOR     ;
0000e802 Voice memo picker, SNOTE_AP TWN     ;/
0000c502 error display JPN-USA-EUR           ;\
0000cf02 error display CHN-KOR-TWN           ;
0000c503 error display JPN-USA-EUR Safe mode ; Non-critical error display
0000cf03 error display CHN-KOR-TWN Safe mode ; (error) (online, etc)
2000c503 error display JPN New3DS Safe mode  ; CTR-N-HEEx
2000c503 error display USA New3DS Safe mode  ;
2000c503 error display EUR New3DS Safe mode  ;
?        error display CHN New3DS Safe mode  ;
2000cf03 error display KOR New3DS Safe mode  ;
N/A      error display TWN New3DS Safe mode  ;/
0000cd02 extrapad JPN-USA-EUR                ;\Circle Pad Pro (extrapad)
0000d502 extrapad CHN-KOR-TWN                ;/CTR-N-HADx test/calibration
0000c602 eShop applet JPN                    ;\eShop applet (mint), used by
0000ce02 eShop applet USA                    ; applications for accessing
0000d602 eShop applet EUR                    ; the eShop, for DLC/etc.
N/A      eShop applet CHN                    ; Also used by the eShop
0000e302 eShop applet KOR                    ; application itself
0000e902 eShop applet TWN                    ;/CTR-N-HAAx
0000bc02 Miiverse, olv JPN                   ;\
0000bd02 Miiverse, olv USA                   ;
0000be02 Miiverse, olv EUR                   ; Miiverse (olv)
?        Miiverse, olv CHN                   ; CTR-N-HAEx
N/A      Miiverse, olv KOR                   ;
?        Miiverse, olv TWN                   ;/
0000f602 Miiverse, memolib or so JPN-USA-EUR ;\Likely the "system library"
?        Miiverse, memolib or so  CHN        ; for Miiverse (memolib)
N/A      Miiverse, memolib or so  KOR        ; CTR-N-HAGA
?        Miiverse, memolib or so  TWN        ;/
00008302 Miiverse-posting applet, solv3 JPN  ;\
00008b02 Miiverse-posting applet, solv3 USA  ; In-app Miiverse-posting
0000ba02 Miiverse-posting applet, solv3 EUR  ; applet (solv3)
?        Miiverse-posting applet, solv3 CHN  ; CTR-N-HAHx
N/A      Miiverse-posting applet, solv3 KOR  ;
?        Miiverse-posting applet, solv3 TWN  ;/
00009502 Cabinet, amiibo Settings JPN        ;\
00009e02 Cabinet, amiibo Settings USA        ;
0000b902 Cabinet, amiibo Settings EUR        ; Cabinet (amiibo Settings)
?        Cabinet, amiibo Settings CHN        ; CTR-N-HA3x
00008c02 Cabinet, amiibo Settings KOR        ;
0000bf02 Cabinet, amiibo Settings TWN        ;/
```

3ds:\title\0004009b Shared Data Archives

```
00010202 Probably Mii-related                ;-RomFS contains "CFL_Res.dat"
00010402 Region Manifest aka area            ;-Mounted as "area:"
00010602 Non-Nintendo TLS Root-CA Certificates;-RomFS contains ".der" files
00011002 Dictionary CHN-CH      CHN          ;\
```

```
00011102 Dictionary TWN-TW      TWN           ;
00011202 Dictionary NL-NL       EUR           ;
00011302 Dictionary EN-GB       EUR           ;
00011402 Dictionary EN-US       USA           ;
00011502 Dictionary FR-FR       EUR           ; Dictionary
00011602 Dictionary FR-CA       USA           ;
00011702 Dictionary DE-regular  EUR           ;
00011802 Dictionary IT-IT       EUR           ;
00011902 Dictionary JA-small-32 JPN           ;
00011a02 Dictionary KO-KO       KOR           ;
00011b02 Dictionary PT-PT       EUR           ;
00011c02 Dictionary RU-regular  EUR           ;
00011d02 Dictionary ES-ES       EUR+USA       ;
00011e02 Dictionary PT-BR       USA           ;/
00012202 Error Strings JPN                    ;\
00012302 Error Strings USA                    ;
00012102 Error Strings EUR                    ; Error Strings
00012402 Error Strings CHN                    ;
00012502 Error Strings KOR                    ;
00012602 Error Strings TWN                    ;/
00013202 Eula JPN                             ;\
00013302 Eula USA                             ; End User blurp
00013102 Eula EUR                             ;
00013502 Eula CHN                             ;/
00014002 System Font JPN-EUR-USA              ;\
00014102 System Font CHN                      ; System Font
00014202 System Font KOR                      ;
00014302 System Font TWN                      ;/
00015202 Rate or so JPN                       ;\
00015302 Rate or so USA                       ; whatever rate
00015102 Rate or so EUR                       ; ... maybe parental control
N/A      Rate or so CHN                       ;     age ratings?
00015502 Rate or so KOR                       ;
0015602  Rate or so TWN                       ;/   ;<-- only 7 letter???
```

3ds:\title\000400db System Data Archives 2

```
00010302 NGWord bad word list                 ;-
00010502 Nintendo Zone hotspot list           ;-
00016102 NVer JPN                             ;\
00016202 NVer USA                             ;
00016302 NVer EUR                             ; NVer ?
00016402 NVer CHN                             ;
00016502 NVer KOR                             ;
00016602 NVer TWN                             ;
20016102 NVer JPN New3DS                      ;
20016202 NVer USA New3DS                      ;
20016302 NVer EUR New3DS                      ;
N/A      NVer CHN New3DS                      ;
20016502 NVer KOR New3DS                      ;
N/A      NVer TWN New3DS                      ;/
00017102 CVer JPN                             ;\
00017202 CVer USA                             ;
00017302 CVer EUR                             ; CVer ?
00017402 CVer CHN                             ;
00017502 CVer KOR                             ;
00017602 CVer TWN                             ;/
```

3ds:\title\00040130 System Modules

```
00001002 sm               ;(Stored in NATIVE_FIRM)
00001003 sm Safe mode     ;(Stored in NATIVE_FIRM Safe mode)
00001102 fs               ;(Stored in NATIVE_FIRM)
00001103 fs Safe mode     ;(Stored in NATIVE_FIRM Safe mode)
00001202 pm               ;(Stored in NATIVE_FIRM)
00001203 pm Safe mode     ;(Stored in NATIVE_FIRM Safe mode)
```

```
00001302 loader           ;(Stored in NATIVE_FIRM)
00001303 loader Safe mode ;(Stored in NATIVE_FIRM Safe mode)
00001402 pxi              ;(Stored in NATIVE_FIRM)
00001403 pxi Safe mode    ;(Stored in NATIVE_FIRM Safe mode)
00001502 Application Manager, AM
00001503 Application Manager, AM Safe mode
20001503 Application Manager, AM Safe mode New3DS
00001602 Camera
20001602 Camera New3DS
00001702 Config, cfg
00001703 Config, cfg Safe mode
20001703 Config, cfg Safe mode New3DS
00001802 Codec
00001803 Codec Safe mode
20001803 Codec Safe mode New3DS
00001a02 DSP
00001a03 DSP Safe mode
20001a03 DSP Safe mode New3DS
00001b02 GPIO
00001b03 GPIO Safe mode
20001b03 GPIO Safe mode New3DS
00001c02 GSP          ;something GPU related?
20001c02 GSP New3DS
00001c03 GSP Safe mode
20001c03 GSP Safe mode New3DS
00001d02 Human Interface Devices HID
00001d03 Human Interface Devices HID Safe mode
20001d03 Human Interface Devices HID Safe mode New3DS
00001e02 i2c
20001e02 i2c New3DS
00001e03 i2c Safe mode
20001e03 i2c Safe mode New3DS
00001f02 MCU
20001f02 MCU New3DS
00001f03 MCU Safe mode
20001f03 MCU Safe mode New3DS
00002002 Microphone MIC
00002102 PDN
00002103 PDN Safe mode
20002103 PDN Safe mode New3DS
00002202 Play time PTM (pedometer, battery manager)
20002202 Play time PTM New3DS (pedometer, battery manager)
00002203 Play time PTM Safe mode
20002203 Play time PTM Safe mode New3DS
00002302 spi
20002302 spi New3DS
00002303 spi Safe mode
20002303 spi Safe mode New3DS
00002402 Network manager, AC
00002403 Network manager, AC Safe mode
20002403 Network manager, AC Safe mode New3DS
00002602 Cecd (StreetPass)
00002702 CSND
00002703 CSND Safe mode
20002703 CSND Safe mode New3DS
00002802 Download Play, DLP
00002902 HTTP
00002903 HTTP Safe mode
20002903 HTTP Safe mode New3DS
00002a02 MP
00002a03 MP Safe mode
00002b02 NDM
00002c02 NIM
00002c03 NIM Safe mode
20002c03 NIM Safe mode New3DS
```

```
00002d02 Low-level wifi manager, NWM
00002d03 Low-level wifi manager, NWM Safe mode
20002d03 Low-level wifi manager, NWM Safe mode New3DS
00002e02 Sockets
00002e03 Sockets Safe mode
20002e03 Sockets Safe mode New3DS
00002f02 SSL
00002f03 SSL Safe mode
20002f03 SSL Safe mode New3DS
00003000 Process9 (in Safe mode and normal NATIVE_FIRM)
00003102 Process Manager, PS
00003103 Process Manager, PS Safe mode
20003103 Process Manager, PS Safe mode New3DS
00003202 friends (Friends list)
00003203 friends (Friends list) Safe mode
20003203 friends (Friends list) Safe mode New3DS
00003302 Infrared, IR
00003303 Infrared, IR Safe mode
20003303 Infrared, IR Safe mode New3DS
00003402 BOSS (SpotPass)
00003502 News (Notifications)
00003702 RO
00003802 act (handles Nintendo Network accounts)
00004002 nfc Old3DS
20004002 nfc New3DS
20004102 mvd New3DS
20004202 qtm New3DS
00008002 NS (Memory-region: "SYSTEM")
00008003 NS (Memory-region: "SYSTEM") Safe mode
20008003 NS (Memory-region: "SYSTEM") Safe mode New3DS
```

3ds:\title\00040138 System Firmware

```
00000001 DevUnit SafeUpdater or so    ;DevUnit, similar to Safe mode_FIRM
00000002 Native Firmware Old3DS       ;NATIVE_FIRM         (Native Firmware)
20000002 Native Firmware New3DS       ;NATIVE_FIRM New_3DS (Native Firmware)
00000003 Safe Mode Old3DS             ;Safe mode_FIRM
20000003 Safe Mode New3DS             ;Safe mode_FIRM New_3DS
00000102 TWL DSi Firmware Old3DS      ;TWL_FIRM         (DSi Firmware)
20000102 TWL DSi Firmware New3DS      ;TWL_FIRM New_3DS (DSi Firmware)
00000202 AGB GBA Firmware Old3DS      ;AGB_FIRM         (GBA Firmware)
20000202 AGB GBA Firmware New3DS      ;AGB_FIRM New_3DS (GBA Firmware)
```

3ds:\data\<ID0>\extdata\00048000:

```
e0000000 System transfer (request transfer if file is present)
f0000001 Camera application NAND JPEG/MPO files, phtcache.bin, UploadData.dat
f0000002 Sound application NAND M4A files
f0000009 SpotPass content storage for notifications
f000000b Miis and Play/Usage Records
f000000c Contains bashotorya.dat and bashotorya2.dat
f000000d Home Menu SpotPass content data storage
f000000e Update notification versionlist.dat (added in 7.0.0-13)
```

3ds:\data\<ID0>\sysdata:

```
System Module Savegames (0001xxxx)...
0001000f reportedly SEEDDB or so
00010011 FS module savedata  ;used for Anti Savegame Restore.
00010015 AM module savedata
00010017 Config savegame
00010022 PTM savegame
00010026 CECD savegame
0001002c NIM savegame
00010032 Friends module savegame
00010034 BOSS module savegame
00010035 News module savegame
```

00010038 Act module savegame
00010040 NFC module savegame
System application and applet savegames (0002xxxx)...
00020082 Home Menu savegame JPN
0002008f Home Menu savegame USA
00020098 Home Menu savegame EUR
00020086 Instruction Manual applet savegame JPN
00020092 Instruction Manual applet savegame USA
0002009b Instruction Manual applet savegame EUR
00020087 Game Notes applet savegame JPN
00020093 Game Notes applet savegame USA
0002009c Game Notes applet savegame EUR
00020088 Old3DS/New3DS Internet Browser savegame JPN
00020094 Old3DS/New3DS Internet Browser savegame USA
0002009d Old3DS/New3DS Internet Browser savegame EUR
0002008d Friend List applet savegame JPN
00020096 Friend List applet savegame USA
0002009f Friend List applet savegame EUR
000200bb Additional savedata t.bin history for the New3DS Browser (ALL)
000200bc olv applet savegame, Miiverse JPN
000200bd olv applet savegame, Miiverse USA
000200be olv applet savegame, Miiverse EUR
000200c5 error applet savegame (ALL)
000200c6 mint applet savegame (ALL)
00020200 System Settings savegame JPN
00020210 System Settings savegame USA
00020220 System Settings savegame EUR
00020202 Activity Log application savegame JPN
00020212 Activity Log application savegame USA
00020222 Activity Log application savegame EUR
00020204 Nintendo 3DS Camera application savegame JPN
00020214 Nintendo 3DS Camera application savegame USA
00020224 Nintendo 3DS Camera application savegame EUR
00020205 Nintendo 3DS Sound application savegame JPN
00020215 Nintendo 3DS Sound application savegame USA
00020225 Nintendo 3DS Sound application savegame EUR
00020207 Mii Maker application savegame JPN
00020217 Mii Maker application savegame USA
00020227 Mii Maker application savegame EUR
00020208 StreetPass Mii Plaza application savegame JPN
00020218 StreetPass Mii Plaza application savegame USA
00020228 StreetPass Mii Plaza application savegame EUR
00020209 eShop application savegame JPN
00020219 eShop application savegame USA
00020229 eShop application savegame EUR
0002020a System Transfer savegame JPN
0002021a System Transfer savegame USA
0002022a System Transfer savegame EUR
0002020b Nintendo Zone savegame JPN
0002021b Nintendo Zone savegame USA
0002022b Nintendo Zone savegame EUR
0002020d Face Raiders savegame JPN
0002021d Face Raiders savegame USA
0002022d Face Raiders savegame EUR
0002020e AR Games savegame JPN
0002021e AR Games savegame USA
0002022e AR Games savegame EUR
000202bf act (NNID settings) application savegame JPN
000202c0 act (NNID settings) application savegame USA
000202c1 act (NNID settings) application savegame EUR
00020231 microSD Management application savegame JPN
00020241 microSD Management application savegame USA
00020251 microSD Management application savegame EUR

3DS Savedata Extdata

This page describes the format and encryption of extdata, "extra data" stored on SD card and NAND, at:

```
nand/data/<ID0>/extdata/<ExtdataID-High>
sdmc/Nintendo 3DS/<ID0>/<ID1>/extdata/<ExtdataID-High>
```

ExtdataID-High is always 00000000 for SD, and always 00048000 for NAND. Regular apps can only mount SD extdata using the same extdataID which is stored in the CXI exheader. Therefore, regular apps which have the exheader extdataID set to zero can't use extdata. This restriction doesn't apply for shared extdata with extdataID high bitmask 48000h stored on NAND. System apps with a certain access right can mount arbitrary extdata. All NAND extdata is shared extdata, while all SD extdata is normal extdata.

All data in this page is little-endian. All "unused / padding" fields can contain uninitialized data unless otherwise specified.

Format

To avoid confusion, the terms device directory / file and virtual directory / file are used with the following meanings:
Device directory / file are the real directory / file stored on SD / NAND that can be seen under path nand/data/<ID0>/extdata/ or sdmc/Nintendo 3DS/<ID0>/<ID1>/extdata/.
Virtual directory / file are directory / file stored inside extdata virtual file system, which can be seen by applications in the mounted extdata archives.

An extdata consists of several device directories and files, which forms a file system consisting of multiple virtual directories and files.

An extdata with ID ExtdataId has the following device files:
.../extdata/<ExtdataID-High>/<ExtdataId-Low>/Quota.dat (optional)
.../extdata/<ExtdataID-High>/<ExtdataId-Low>/<SubDirID>/<SubFileID>

Note:
All device files are DIFF containers. All format description below is about the inner content of the containers. Please unwrap these files first according to the DIFF format description before reading them using the extdata format description below.
Quota.dat is only observed existing for NAND shared extdata.
<SubDirID> and <SubFileID> are 8-digit hex strings.
Device file with SubDirID = SubFileID = 00000000 doesn't exist. Other ID combinations can exists.
Device file with SubDirID = 00000000 and SubFileID = 00000001 is the VSXE metadata file and must exist.
Other files, besides Quota.dat and 00000000/00000001, are normal sub files, are these device files one-to-one correspond to virtual files. They contain raw virtual file data in the DIFF inner content.
SubDirID = 00000000 is usually the only one device directory that can be seen. See #Device Directory Capacity for more information.

Quota File

The inner data of Quota.dat is 48h bytes with the following format. The exact function of this file is unclear.

```
000h 4       Magic "QUOT"
004h 4       Magic 30000h
008h 4       1000h, block size?
00Ch 4       Always 126. Probably device directory capacity. See the
               #Device Directory Capacity more information.
010h 38h     unknown
```

Device Directory Capacity

A device directory in an extdata (a <SubDirID> directory) seems to have a maximum number of device files it

can contain. For SD extdata, this maximum number seems to be hard-coded as 126. For NAND extdata, the number is probably indicated by a field in Quota.dat, which is, again, always 126 as observed. 3DS FS tries to put all device files in the device directory 00000000 if possible, and only when more than 126 files needed to add, a second device directory 00000001 and so on are created. However, few extdata have such amount of files to store, so the behavior lacks of use cases to confirm.

The number 126 is probably from some kind of other capacity of 128 with "." and ".." entries reserved. It is theorized that this is to keep a FAT directory table, with 20h bytes for each entry, in one 1000h cluster. The motivation is unclear.

VSXE File System Metadata

The inner data of 00000000/00000001 consists of the following components

```
VSXE header
Directory Hash Table
File Hash Table
File Allocation Table
Data region
Directory Entry Table
File Entry Table
```

VSXE Header

```
 000h 4    Magic "VSXE"
 004h 4    Magic 30000h
 008h 8    File system Information offset (138h)
 010h 8    Image size in blocks
 018h 4    Image block size
 01Ch 4    Padding
 020h 8    Unknown
 028h 4    'Action' made on most recently mounted Extdata image
 02Ch 4    Unknown
 030h 4    D of most recently mounted Extdata image
 034h 4    Unknown
 038h 100h Mount path, from most recently mounted Extdata image
Below is File system Information, which is assumed following the same layout
as [[Savegames#SAVE Header, uh, what?
 138h 4    Unknown
 13Ch 4    Data region block size
 140h 8    Directory hash table offset
 148h 4    Directory hash table bucket count
 14Ch 4    Padding
 150h 8    File hash table offset
 158h 4    File hash table bucket count
 15Ch 4    Padding
 160h 8    File allocation table offset
 168h 4    File allocation table entry count
 16Ch 4    Padding
 170h 8    Data region offset
 178h 4    Data region block count (=File allocation table entry count)
 17Ch 4    Padding
 180h 4    Directory entry table starting block
 184h 4    Directory entry table block count
 188h 4    Maximum directory count
 18Ch 4    Padding
 190h 4    File entry table starting block
 194h 4    File entry table block count
 198h 4    Maximum file count
 19Ch 4    Padding
```

All "offsets" are relative to the beginning of VSXE image. All "starting block index" are relative to the beginning of data region.

File Allocation Table & Data Region

These function in the same way as the file allocation in savegames. However, the only two "files" allocated in the data region are the directory entry table and file entry table, so the data region is usually pretty small, and the file allocation table is unchanged once created and has no free blocks. Thus, the offset and size of directory / file entry table can be found directly by offset = entry_table_starting block * data_region_block_size + data_region_offset and size = entry_table_block_count * data_region_block_size

Directory Hash Table & File Hash Table
These function in the same way as those in savegames

Directory Entry Table
This functions in the same way as the one in savegames. It lists all virtual directories in this extdata.

File Entry Table
This functions in a similar way as the one in savegames. It lists all virtual files in this extdata. However, the format of a (non-dummy) file entry is a little bit modified:

```
00h 4   Parent directory index in directory entry table
04h 16  ASCII file name
14h 4   Next sibling file index. 0 if this is the last one
18h 4   Padding
1Ch 4   First block index in data region (always 80000000h because unused)
20h 8   File size Unique identifier
28h 4   Padding?
2Ch 4   Index of the next file in the same hash table bucket (0=None)
```

Each non-dummy file entry corresponds to a device file. The path to the device file is generated by the following computation:

```
// See previous section about this capacity
const uint32_t device_dir_capacity = 126;

// entry index is the index in the file entry table, with the first dummy
// entry as index = 0, which is never used for a real file.
// file_index = 1 is reserved for the VSXE Filesystem Metadata itself, so
// real files started from file_index = 2.
uint32_t file_index = entry_index + 1;

uint32_t SubDirID = file_index / device_dir_capacity;
uint32_t SubFileID = file_index % pdevice_dir_capacity;

char extdata_path[...]; // ".../extdata/<ExtdataID-High>/<ExtdataId-Low>"
char device_path[...]; // output path
sprintf(device_path, "%s/%08x/%08x", extdata_path, SubDirID, SubFileID);
```

When mounting extdata, the unique identifier is used to match the ID stored in subfile's DIFF header. If the ID doesn't match, mounting will fail.

Virtual File System Structure
When extdata is created, these are always created regardless of whether the title actually uses them.

```
/icon  This virtual file contains the extdata icon displayed in data
         management. This icon can only be written to by titles when
         creating extdata, titles would have to recreate extdata to
         change the icon. This file can't be read directly, instead it
         is read via FS:ReadExtSaveDataIcon.
/user/ This virtual directory contains the title's actual extdata files.
/boss/ This virtual directory can contain SpotPass content. SpotPass
         content can only be downloaded to this /boss virtual directory.
```

User extdata and SpotPass extdata use separate mount points at /user and /boss. Therefore one mount can't access the other virtual directory, and also can't access /icon (the title's SpotPass extdata can be mounted by the title itself, if it uses SpotPass).

Other optional but notable directories include:

```
/user/ExBanner This virtual directory can optionally store extended banners.
  When this is available, this banner is displayed instead of the CXI ExeFS
  banner. COMMON.bin stores the common exbanner, while <regionlang_code>.bin
  stores an optional separate region/language specific banner
  (regionlang_code can be "JPN_JP", "USA_EN", etc).
```

SD Extdata

Usually the ExtdataID low is in the format '00<Unique ID>'

```
JPN      USA      EUR      Description     Extdata images
00000082 0000008f 00000098 Home Menu extdata, this contains home-menu
                             savedata and cached icons for applications.
00000200 00000210 00000220 System Settings extdata added with 2.0.0-2.
00000207 00000217 00000227 Mii Maker, contains an ExBanner      cleartext
00000208 00000218 00000228 Streetpass Mii Plaza                 11 mb big!
00000209 00000219 00000229 eShop, contains store music in AAC format.
0000020b 0000021b 0000022b Nintendo Zone
0000020d 0000021d 0000022d Face Raiders, likely contains an ExBanner
000002cc 000002cd 000002ce Home Menu theme
?        000004aa 000004ab Nintendo Video Extra Data
```

This is where the video files are stored, and includes the thumbnail, the description, and possibly some checksum info in each video file stored in the extdata images. There are always 9 files within the subdirectory "00000000" of this folder, even without any videos downloaded. The files are "00000001" - "00000009", and "00000003" - "00000008" have the same filesize of 50.7 MB. It is possible to restore the older videos by overwriting all the files within this directory. Provided of course you have made a backup of the files before hand, by copying all the files within this directory to your computer. As far I'm aware its not possible to mix and match the files in order to get certain videos in one grouping, ie. having all 3 Zelda orchestral recordings in one group of 4 Nintendo videos.

```
00000306 00000308 00000307 Mario Kart 7
0000030b 0000030d 0000030c Nintendogs + Cats
00000326 00000326 00000326 Pok,dex 3D
00000305 0000032d 0000033c Super Street Fighter IV 3D
00000328 00000358 0000033b Ridge Racer 3D
?        0000034d 00000402 Samurai Warriors Chronicles
?        0000034f 0000038a Dead or Alive Dimensions
00000481 N/A      N/A      Monster Hunter Tri G (Download-Quests)
?        00000517 00000518 Swapnote
0000055d 0000055d 0000055d P-Letter X, P-Letter Y
?        00000725 00000724 Ambassador Certificate
?        ?        000007af New Super Mario Bros. 2
?        00000863 00000864 Animal Crossing: New Leaf
?        00000a85 00000a86 Professor Layton and the Miracle Mask
                             Professor Layton and the Azran Legacy
                             German Version ExtdataID is 00000a87
?        ?        00000b4f Fullblox / Crashmo
?        ?        00000ba9 Pok,mon Mystery Dungeon: Gates to Infinity
?        ?        00000c24 Denpa men
00000c73 00000c73 00000c73 Save Data Transfer Tool
?        ?        00000d9a Donkey Kong CountryÖ
                                Returns 3D: Trailer
?        ?        00000ea6 Etrian Odyssey IV
?        00000edf 00000ee0 Super Smash Bros. for Nintendo 3DS
?        00000f14 00000f1e Phoenix Wright: Ace Attorney - Dual Destinies
?        00001007 00001005 Professor Layton vs Phoenix Wright: Ace Attorney
?        ?        00001062 Nintendo Pocket Football Club
?        ?        0000111c Yoshi's New Island
?        ?        00001131 Fantasy Life
000011c5 000011c5 000011c5 Pok,mon Omega Ruby, Pok,mon Alpha Sapphire
?        ?        000012ca Mario vs. Donkey Kong: Tipping Stars
?        ?        00001499 Korg DSN-12
?        ?        000014f2 Animal Crossing: Happy Home Designer
000014d1 000014d1 000014d1 Home Menu badge
?        ?        00001632 Fullblox / Stretchmo
?        ?        00001646 Pok,mon Rumble World
```

```
00001648 00001648 00001648  Pok,mon Sun, Pok,mon Moon
0000165c 0000165c 0000165c  Home Menu saved theme layouts
?        ?        00001678  Yo-kai Watch
?        ?        000018fa  Phoenix Wright: Ace Attorney - Spirit of Justice
?        ?        0000198f  Animal Crossing: New Leaf - Welcome amiibo
?        ?        00001a05  Super Mario Maker
?        ?        00001a2e  Swapdoodle
```

NAND Shared Extdata

```
e0000000 Request system transfer (if present)
f0000001 Camera application NAND JPEG/MPO files, phtcache.bin, UploadData.dat
f0000002 Sound application NAND M4A files
f0000009 SpotPass content storage for notifications
f000000b Miis and Play/Usage Records
           Contains idb.dat, idbt.dat, gamecoin.dat, ubll.lst, CFL_DB.dat,
           and CFL_OldDB.dat. These files contain cleartext Miis and some
           data relating (including cached ICN data) to Play/Usage Records
f000000c Contains bashotorya.dat and bashotorya2.dat
f000000d Home Menu SpotPass content data storage
f000000e Update notification versionlist.dat (added in 7.0.0-13)
```

Shared Extdata "f000000b" gamecoin.dat

```
000h 4  Magic number: 4F00h
004h 2  Total Play Coins
006h 2  Total Play Coins obtained on the date stored below. When the below
          date does not match the current date, this field is reset to zero,
          then the date (and other fields) are updated. Once this value
          is >=10, no more Play Coins can be obtained until the current date
          changes.
008h 4  Total step count at the time a new Play Coin was obtained.
00Ch 4  Step count for the day the last Play Coin was obtained, for that
          day's step count (same as the step count displayed by home-menu
          when this file was updated).
010h 2  Year
012h 1  Month
013h 1  Day
```

The above date stores the last time new Play Coin(s) were obtained. The contents of this file is updated by home-menu. PTM:GetTotalStepCount is not checked constantly, after home-menu boot this is only checked when waking from sleep-mode. Each time home-menu updates the contents of this file, home-menu will set the Play Coin total to 300 if it's higher than the 300 Play Coin limit.

Home Menu loads this file / opens this archive during startup. When accessing this file fails, like when the file/archive is corrupted (or at least on older system-versions), the result is a brick due to Home Menu using svcBreak. Yellows8 bricked a 3DS this way due to corruption via invalid FSFile:Write flush flags. When opening this extdata archive ("f000000b") fails, Home Menu executes svcBreak.

Shared Extdata "f000000b" ubll.lst

List of blocked users.
Empty space is filled with 0Ch-long sequences of 00 00 ... 07

Tools

3ds-save-tool - Extract/verifies extdata

3DS Savedata Savegames

This page describes the format and encryption of savegames contained in gamecards, SD and NAND. You can find savegames from various 3DS games on the Games page.

This page does not describe DISA container format, which all savegames use as wrappers.

All data in this page is little-endian unless otherwise specified. All "unused / padding" fields can contain uninitialized data unless otherwise specified.

Overview
Savegames are stored in DISA container format (follow this link for the container format description). It forms a file system inside the inner content of the container. In this page only the inner file system format of the content is described.

Unlike SD and NAND savegames, gamecard savegames has additional encryption + wear leveling layer. They are described in the following sections.

Gamecard savegame Encryption

Repeating CTR Fail
On the 3DS savegames are stored much like on the DS, that is on a FLASH chip in the gamecart. On the DS these savegames were stored in plain-text but on the 3DS a layer of encryption was added. This is AES-CTR, as the contents of several savegames exhibit the odd behavior that xor-ing certain parts of the savegame together will result in the plain-text appearing.

The reason this works is because the stream cipher used has a period of 512 bytes. That is to say, it will repeat the same keystream after 512 bytes. The way you encrypt with a stream cipher is you XOR your data with the keystream as it is produced. Unfortunately, if your streamcipher repeats and you are encrypting a known plain-text (in our case, zeros) you are basically giving away your valuable keystream.

So how do you use this to decrypt a savegame on a 3DS? First off, you chunk up the savegame into 512 byte chunks. Then, you bin these chunks by their contents, discarding any that contain only FF. Now look for the most common chunk. This is your keystream. Now XOR the keystream with your original savegame and you should have a fully decrypted savegame. XOR with the keystream again to produce an encrypted savegame.

Savegame keyY
All gamecard and SD savegames are encrypted with AES-CTR. The base CTR for gamecard savegames is all-zero. The gamecard savegame keyslots' keyY (these savegame keyslots use the hardware key-generator) is unique for each region and for each game. The NCSD partition flags determine the method used to generate this keyY. When the save NCSD flags checked by the running NATIVE_FIRM are all-zero, the system will use the repeating CTR, otherwise a proper CTR which never repeats within the image is used.

The AES-CMAC (which uses a hardware key-generator keyslot, as mentioned above) at the the beginning of the savegame must match the calculated CMAC using the DISA/DIFF data, otherwise the savegame is considered corrupted (see below).

When all of the flags checked by the running NATIVE_FIRM are clear, the keyY (original keyY method used with saves where the CTR repeats within the image) is the following:

```
  00h 8     First 8-bytes from the plaintext CXI accessdesc signature.
  08h 4     u32 CardID0 from gamecard plaintext-mode command 0x90,
              Process9 reads this with the NTRCARD hw. The actual cmdID used
              by Process9 is different since Process9 reads it with the
              gamecard in encrypted-mode.
  0Ch 4     u32 CardID1 from gamecard plaintext-mode command 0xA0,
              Process9 reads this with the NTRCARD hw. The actual cmdID used
              by Process9 is different since Process9 reads it with the
              gamecard in encrypted-mode.
```

2.0.0-2 Hashed keyY and 2.2.0-4 Savegame Encryption

When certain NCSD partition flags are set, a SHA-256 hash is calculated over the data from the CXI (same data used with the original plain keyY), and the 0x40-bytes read from a gamecard command (this 0x40-byte data is also read by GetRomId, which is the gamecard-uniqueID). The first 0x10-bytes from this hash is used for the keyY. When flag[7] is set, the CTR will never repeat within the save image, unlike the original CTR-method. All games which had the retail NCSD image finalized after the 2.2.0-4 update (and contain 2.2.0-4+ in the System update partition), use this encryption method.

This keyY generation method was implemented with 2.0.0-2 via NCSD partition flag[3], however the proper CTR wasn't implemented for flag[7] until 2.2.0-4. The hashed keyY flag[3] implemented with 2.0.0-2 was likely never used with retail gamecards.

6.0.0-11 Savegame keyY
6.0.0-11 implemented support for generating the savegame keyY with a new method, this method is much more complex than previous keyY methods. This is enabled via new NCSD partition flags, all retail games which have the NCSD image finalized after the 6.0.0-11 release (and 6.0.0-11+ in the system update partition) will have these flags set for using this new method.

A SHA-256 hash is calculated over the same data used with the above hashed keyY method, after hashing the above data the following data is hashed: the CXI programID, and the ExeFS:/.code hash from the decrypted ExeFS header. An AES-CMAC (the keyslot used for this uses the hardware key-scrambler) is then calculated over this hash, the output CMAC is used for the savegame keyY.

The keyY used for calculating this AES-CMAC is initialized while NATIVE_FIRM is loading, this keyY is generated via the RSA engine. The RSA slot used here is slot0 (key-data for slot0 is initialized by bootrom), this RSA slot0 key-data is overwritten during system boot. This RSA slot0 key-data gets overwritten with the RSA key-data used for verifying RSA signatures, every time Process9 verifies any RSA signatures except for NCCH accessdesc signatures. Starting with 7.0.0-13 this key-init function used at boot is also used to initialize a separate keyslot used for the new NCCH encryption method.

This Process9 key-init function first checks if a certain 0x10-byte block in the 0x01FF8000 region is all-zero. When all-zero it immediately returns, otherwise it clears that block then continues to do the key generation. This is likely for supporting launching a v6.0+ NATIVE_FIRM under this FIRM.

Gamecard wear leveling
The 3DS employs a wear leveling scheme on the savegame FLASH chips (only used for CARD1 gamecards). This is done through the usage of blockmaps and a journal. The blockmap is located at offset 0 of the flash chip, and is immediately followed by the journal. The initial state is dictated by the blockmap, and the journal is then applied to that.

First, there are 8 bytes whose purposes are currently unknown. Then comes the actual blockmap. The blockmap structure is simple:

```
  struct header_entry {
    uint8_t phys_sec  ;when bit7=1: block has chksums (else chksums are all 0)
    uint8_t alloc_cnt
    uint8_t chksums[8]
  } __attribute__((__packed__));
```

There's one entry per sector, counting from physical sector 1 (sector 0 contains the blockmap/journal).
The 2 bytes that follow the blockmap are the CRC16 (with initial value FFFFh) of the first 8 bytes and the blockmap.

Then comes the journal. The journal structure is as follows:

```
  struct sector_entry {
    uint8_t virt_sec         ;Mapped to sector
    uint8_t prev_virt_sec    ;Physical sector previously mapped to
    uint8_t phys_sec         ;Mapped from sector
    uint8_t prev_phys_sec    ;Virtual sector previously mapped to
```

```
    uint8_t phys_realloc_cnt ;Amount of times physical sector has been remapped
    uint8_t virt_realloc_cnt ;Amount of times virtual sector has been remapped
    uint8_t chksums[8]
  } __attribute__((__packed__));

  struct long_sector_entry{
    struct sector_entry sector
    struct sector_entry dupe
    uint32_t magic           ;With magic being a constant 080D6CE0h.
  }__attribute__((__packed__));
```

The checksums in the blockmap/journal entries work as follows:
each byte is the checksum of an encrypted 200h bytes large block,
to calculate the checksum, a CRC16 of the block (with initial value FFFFh) is calculated, and the two bytes of the CRC16 are XORed together to produce the 8bit checksum

Components and partitions
A savegame, after unwrapping the DISA container, consists of the following components:

```
SAVE header
directory hash table
file hash table
file allocation table
directory entry table
file entry table
data region
```

A DISA container can have one or two partitions, and correspondingly a savegame has two possible layouts. The layout is determined by the parameter duplicate data passed in FS:FormatSaveData or FS:CreateSystemSaveData.

Layout for duplicate data = true
The DISA container only has one partition which is always configured as external IVFC level 4 disabled (see DISA format for details). All components are stored in this partition as

```
SAVE header at the beginning
directory hash table
file hash table
file allocation table
data region
directory entry table is allocated inside data region
file entry table as well
all file data is also allocated here
```

In this layout, all data is duplicated by DISA's DPFS tree, which is what the parameter duplicate data implies.

Layout for duplicate data = false
The DISA container has two partitions. Partition A is always configured as external IVFC level 4 disabled, and partition B is configured as it enabled. Components are stored among the two partitions as

```
Partition A
SAVE header at the beginning.
directory hash table
file hash table
file allocation table
directory entry table
file entry table
Partition B
used as data region entirely, and only has file data allocated.
```

In this layout, all file system metadata is duplicated by partition A DPFS tree, but file data is not as partition B has external IVFC level 4.

SAVE Header
The SAVE header defines the rest components of the savegame. All "offsets" in the table below are relative to

the beginning of partition A (inner content), while all "starting block index" are relative to the beginning of data region.

```
  000h 4   Magic "SAVE"
  004h 4   Magic 40000h
  008h 8   File system Information offset (20h)
  010h 8   Image size in blocks
  018h 4   Image block size
  01Ch 4   Padding
          Below is File system Information
  020h 4   Unknown
  024h 4   Data region block size
  028h 8   Directory hash table offset
  030h 4   Directory hash table bucket count
  034h 4   Padding
  038h 8   File hash table offset
  040h 4   File hash table bucket count
  044h 4   Padding
  048h 8   File allocation table offset
  050h 4   File allocation table entry count
  054h 4   Padding
  058h 8   Data region offset (if no partition B)
  060h 4   Data region block count (=File allocation table entry count)
  064h 4   Padding
  068h 8   If partition B exists: directory entry table offset;
           otherwise: u32 directory entry table starting block index, and
                      u32 directory entry table block count
  070h 4   Maximum directory count
  074h 4   Padding
  078h 8   If partition B exists: file entry table offset;
           otherwise: u32 file entry table starting block index, and
                      u32 file entry table block count
  080h 4   Maximum file count
  084h 4   Padding
```

The file/directory bucket count & maximum count are specified by the parameters of FS:FormatSaveData or FS:CreateSystemSaveData.

When partition B doesn't exist, directory & file entry tables are allocated in the data region, and while be marked allocated in file allocation table as if they are two normal files. However, only continuous allocation has been observed, so directly reading block_count * block_size bytes from data_region + starting_block_index * block_size should be safe. See the section #File Allocation Table below for more information.

Directory Entry Table

The directory entry table is an array of the entry type shown below. It describes the directory hierarchy of the file system.

```
  000h 4   Parent directory index. 0 for root
  004h 16  ASCII directory name in. All zero for root
  014h 4   Next sibling directory index. 0 if this is the last one
  018h 4   First subdirectory index. 0 if not exists
  01Ch 4   First file index in file entry table. 0 for empty directory
  020h 4   Padding / zero?
  024h 4   Index of the next directory in the same hash table bucket. 0 if
             this is the last one
```

There are also some dummy entries in the array:

```
  000h 4   Current Total entry count
  004h 4   Maximum entry count = maximum directory count + 2
  008h 28  Padding / All zero
  024h 4   Index of the next dummy entry. 0 if this is the last one
```

The 0-th entry of the array is always a dummy entry, which functions as the head of the dummy entry linked list. The 1-st entry of the array is always the root. Therefore maximum entry count is two more than maximum directory count. Dummy entries are left there when deleting directories, and reserved for future use.

File Entry Table

The file entry table is an array of the entry type shown below. It contains information for each file.

```
000h 4   Parent directory index in directory entry table
004h 16  ASCII file name
014h 4   Next sibling file index. 0 if this is the last one
018h 4   Padding
01Ch 4   First block index in data region (80000000h=None)
020h 8   File Size
028h 4   Padding?
02Ch 4   Index of the next file in the same hash table bucket. 0 if
           this is the last one
```

Like directory entry table, file entry table also has some dummy entries:

```
000h 4   Current total entry count
004h 4   Maximum entry count = maximum file count + 1
008h 36  Padding / All zero
02Ch 4   Index of the next dummy entry. 0 if this is the last one
```

The 0-th entry of the array is always a dummy entry, which functions as the head of the dummy entry linked list. Therefore maximum entry count is one more than maximum file count. Dummy entries are left there when deleting files, and reserved for future use.

Directory Hash Table & File Hash Table

This is a u32 array of size = bucket count, each of which is an index to the directory / file entry table. The directory / file name is hashed and its entry index is put to the corresponding bucket. If there is already a directory/file entry in the bucket, then it appends to the linked list formed by Index of the next directory/file in the same hash table bucket field in the directory/file entry table. i.e. this is a hash table using separate chaining with linked lists

The hash function takes the parent index and the name as key. The function is equivalent to

```
uint32_t GetBucket(
    char name[16], // takes all 16 bytes including trailing zeros
    uint32_t parent_dir_index,
    uint32_t bucket_count
) {
    uint32_t hash = parent_dir_index ^ 0x091A2B3C;
    for (int i = 0; i < 4; ++i) {
        hash = (hash >> 1) | (hash << 31);
        hash ^= (uint32_t)name[i * 4]
        hash ^= (uint32_t)name[i * 4 + 1] << 8
        hash ^= (uint32_t)name[i * 4 + 2] << 16
        hash ^= (uint32_t)name[i * 4 + 3] << 24
    }
    return hash % bucket_count;
}
```

File Allocation Table

The file allocation table is an array of a 8-byte entry shown below. The array size is actually one larger than the size recorded in the SAVE header. Each entry corresponds to a block in the data region (the block size is defined in SAVE header). However, the 0th entry corresponds to nothing, so the corresponding block index is off by one. e.g. entry 31 in this table corresponds to block 30 in the data region.

```
000h 4       bit[0:30]: Index U; bit[31]: Flag U
004h 4       bit[0:30]: Index V; bit[31]: Flag V
```

Entries in this table forms several chains, representing how blocks in the data region should be linked together. However, unlike normal FAT systems, which uses chains of entries, 3DS savegames use chain of nodes. Each node spans one or multiple entries.

One node spanning n entries starting from FAT[k] is in the following format:

```
FAT[k + 0]:
  Index_U = index of the first entry of the previous node. 0 if this is
            the first node.
  Index_V = index of the first entry of the next node. 0 if this is the
            last node.
  Flag_U set if this is the first node.
  Flag_V set if this node has multiple entries.

FAT[k + 1]:
  Index_U = k (the first entry index of this node)
  Index_V = k + n - 1 (the last entry index of this node)
  Flag_U always set
  Flag_V always clear

FAT[k + 2] ~ FAT[k + n - 2]:
  All these entries are uninitialized

FAT[k + n - 1]:
  Index_U = k
  Index_V = k + n - 1
  Flag_U always set
  Flag_V always clear
  (Same values as FAT[k + 1])
```
Note: all indices above are entry indices (block index + 1)

All free blocks that are not allocated to any files also form a node chain in the allocation table. The head index of this "free chain" is recorded in FAT[0].Index_V. Other fields of FAT[0] are all zero
Here is an example: [1]

Initialization
When a save FLASH contains all xFFFF blocks it's assumed uninitialized by the game cartridges and it initializes default data in place, without prompting the user. The FFFFFFFFh blocks are uninitialized data. When creating a non-gamecard savegame and other images/files, it's initially all FFFFFFFFh until it's formatted where some of the blocks are overwritten with encrypted data.

Whatever --Elisherer
I got a new game SplinterCell3D-Pal and I downloaded the save and it was 128KB of FFh, except the first 10h bytes which were the letter 'Z' (uppercase) --Elisherer 22:41, 15 October 2011 (CEST)
Fun Facts:
If you have facts that you found out by looking at the binary files please share them here:
From one save to another the game backups the last files that were in the partition and the entire image header in "random" locations.. --Elisherer 22:41, 15 October 2011 (CEST)

Tools
3dsfuse supports reading and modifying savegames. In the mounted FUSE filesystem, the /output.sav is the raw FLASH save-image. When the save was modified, a separate tool to update the CMAC must be used with /clean.sav, prior to writing output.sav to a gamecard.
3DSExplorer supports reading of savegames, it doesn't support reading the new encrypted savegames and maybe in the future it will support modifying (some of the modyfing code is already implemented).
wwylele's 3ds-save-tool supports extracting files from savegames and extdata. It properly reconstructs data from the DPFS tree and extracts files in directories hierarchy.
3dsfuse-ex similar to 3dsfuse, but supports savegame inner FS, proper DPFS handling, and automatic CMAC update. Still WIP.

3DS Savedata DISA and DIFF

This page describes DISA and DIFF format as the underlying container of Savegames, Extdata and Title Database. For further format specification of the inner data, please refer to their own page.

All data in this page is little-endian. All "unused / padding" fields can contain uninitialized data unless otherwise specified.

Overview
DISA and DIFF are two container formats. They are very similar and are used for various purposes in 3DS. Here is a summary table of their usage, the CMAC type and the AES key slot used (the meaning of these is explained in the next section):

Usage	Media	Format	CMAC type	CMAC Keyslot
Savegames	Gamecard	DISA	CTR-SAV0	19h
Savegames	SD	DISA	CTR-SIGN	30h
System SaveData	NAND	DISA	CTR-SYS0	30h
Private Extdata	SD	DIFF	CTR-EXT0	30h
Shared Extdata	NAND	DIFF	CTR-EXT0	30h
Title Database	SD	DIFF	CTR-9DB0	30h
Title Database	NAND	DIFF	CTR-9DB0	0Bh

Encryption
DISA and DIFF formats don't have their own encryption specification. They follow the encryption method defined by their media:
Gamecard savegames have special wear leveling + encryption layers. See Savegames for detail.
Files on SD follow the general SD filesystem encryption rule.
Files on NAND are in cleartext, after decrypting the NAND partition encryption.

Format
A DISA / DIFF file consists of the following components:

```
100h-byte AES CMAC
100h-byte Header
Secondary partition table
Contains 1-2 partition descriptors, depending on the number of partitions
Primary partition table
Same layout as the secondary one
Partition A
Partition B (optional)
can only exist for DISA.
```

AES CMAC
The AES CMAC is located at the beginning of the DISA / DIFF image, and it is 10h long. The rest F0h bytes before the header are unused.
The key used for the AES CMAC is generated by the hardware key engine. See the keyslot it uses in the table above.
The data being authenticated by the AES CMAC is a 20h-byte SHA-256 hash of a data block. The data block has different content formats among CMAC types. All types of data block contain a copy or a hash of the header, which is the start of the the rest of the verification chain, so the AES CMAC effectively authenticates the whole save image. Each type of data block is explained below.

CTR-SAV0
This CMAC type is used for gamecard savegames. It is 28h-byte long.

```
000h 8    Magic "CTR-SAV0"
008h 20h  SHA-256 of the following 108h-byte block
028h 8    Magic "CTR-NOR0"
030h 100h Copy of the DISA header
```

CTR-SIGN

This CMAC type is used for SD savegames. It is 30h-byte long.

```
000h 8    Magic "CTR-SIGN"
008h 8    Title ID
010h 20h  SHA-256 of the following 108h-byte block
030h 8    Magic "CTR-SAV0"
038h 100h Copy of the DISA header
```

CTR-SYS0

This CMAC type is used for NAND system save. It is 110h-byte long.

```
000h 8    Magic "CTR-SYS0"
008h 8    Save ID. The higher word is always zero
010h 100h Copy of the DISA header
```

CTR-EXT0

This CMAC type is used for extdata. It is 11Ch-byte long.

```
000h 8    Magic "CTR-EXT0"
008h 8    Extdata ID
010h 4    0 for Quota.dat, 1 otherwise
014h 4    ID in the device file name                          ;\zero for
018h 4    ID in the device directory name that the file is in ;/Quota.dat
01Ch 100h Copy of the DIFF header
```

CTR-9DB0

This CMAC type is used for title database. It is 10Ch-byte long.

```
000h 8    Magic "CTR-9DB0"
008h 4    Database ID. Each .db file has its own ID
00Ch 100h Copy of the DIFF header
```

Header

The header located at offset 100h defines the rest components of the file (partitions and their tables). All offsets in the header are relative to the beginning of the DISA/DIFF file, except for partition descriptor offsets, which are relative to the beginning of the (active) partition table. DISA and DIFF have different header format.

DISA header

```
000h 4   Magic "DISA"
004h 4   Magic 40000h
008h 4   Partition count, 1 or 2
00Ch 4   Padding
010h 8   Secondary partition table offset
018h 8   Primary partition table offset
020h 8   Partition table size
028h 8   Partition A descriptor offset in the partition table
030h 8   Partition A descriptor size
038h 8   Partition B descriptor offset in the partition table
040h 8   Partition B descriptor size
048h 8   Partition A offset
050h 8   Partition A size
058h 8   Partition B offset
060h 8   Partition B size
068h 1   Active partition table, 0 = primary, 1 = secondary
069h 3   Padding
06Ch 20h SHA-256 over the active partition table
08Ch 74h Unused
```

Note:

When the partition count is 1, there is no partition B and all of its related fields are zero.

DIFF header

```
000h 4   Magic "DIFF"
004h 4   Magic 30000h
008h 8   Secondary partition table/descriptor offset
010h 8   Primary partition table/descriptor offset
```

```
018h 8   Partition table/descriptor size
020h 8   Partition (A) offset
028h 8   Partition (A) size
030h 4   Active partition descriptor, 0 = primary, 1 = secondary
034h 20h SHA-256 over the active partition table/descriptor
054h 8   Unique identifier
05Ch A4h Unused, might contain leftover data
```

Note:
Since DIFF can only contain one partition, a partition table can only have one partition descriptor, so they become the same concept here.
See Extdata for its usage of the unique identifier field. For title database files, this field is zero.

Partition table & partition descriptor
There are two partition tables, but only one of them is active. When operating on a DISA / DIFF file, 3DS FS alternately activate one of the two tables, presumably for data backup or atomic file writing. A newly created DISA / DIFF file may have entirely uninitialized data in the inactive partition table.

One partition table contains one or two partition descriptors , each of which describes the layout of one partition. A partition descriptor contains the following components:

```
DIFI header
IVFC descriptor
DPFS descriptor
Partition master hash
A four-byte padding
```

DIFI header
The DIFI header locates at the beginning of a partition descriptor. This header defines the rest components of the partition descriptor (IVFC descriptor, DPFS descriptor and partition master hash). All offsets are relative to the beginning of the partition descriptor, except for External IVFC level 4 offset, which is relative to the beginning of the partition.

```
000h 4   Magic "DIFI"
004h 4   Magic 10000h
008h 8   IVFC descriptor offset
010h 8   IVFC descriptor size
018h 8   DPFS descriptor offset
020h 8   DPFS descriptor size
028h 8   Partition hash offset
030h 8   Partition hash size
038h 1   If none zero, enable external IVFC level 4.
039h 1   DPFS tree level 1 selector
03Ah 2   Padding
03Ch 8   External IVFC level 4 offset, zero if external IVFC level 4 disabled
```

Note:
The meaning of fields after 38h are explained in the section #Partition

IVFC descriptor
This header defines each level of IVFC tree (explained in the section #Partition). All offsets are relative to the beginning of DPFS level 3.

```
000h 4   Magic "IVFC"
004h 4   Magic 20000h
008h 8   Master hash size = partition master hash size in DIFI header
010h 8   IVFC level 1 offset
018h 8   IVFC level 1 size
020h 4   IVFC level 1 block size in log2
024h 4   Padding
028h 8   IVFC level 2 offset
030h 8   IVFC level 2 size
038h 4   IVFC level 2 block size in log2
03Ch 4   Padding
040h 8   IVFC level 3 offset
```

```
048h 8   IVFC level 3 size
050h 4   IVFC level 3 block size in log2
054h 4   Padding
058h 8   IVFC level 4 offset (unused if external IVFC level 4 enabled)
060h 8   IVFC level 4 size
068h 4   IVFC level 4 block size in log2
06Ch 4   Padding
070h 8   IVFC descriptor size? The value is usually 78h
```

DPFS descriptor
This header defines each level of DPFS tree (explained in the section #Partition). All offsets are relative to the beginning of the partition.

```
000h 4   Magic "DPFS"
004h 4   Magic 10000h
008h 8   DPFS level 1 offset
010h 8   DPFS level 1 size
018h 4   DPFS level 1 block size in log2 (unused?)
01Ch 4   Padding
020h 8   DPFS level 2 offset
028h 8   DPFS level 2 size
030h 4   DPFS level 2 block size in log2
034h 4   Padding
038h 8   DPFS level 3 offset
040h 8   DPFS level 3 size
048h 4   DPFS level 3 block size in log2
04Ch 4   Padding
```

Partition master hash
This is a SHA-256 hash list over IVFC level 1. See #IVFC tree for explanation.

Partition
A partition can have two types of layout. This is determined by the field DIFF+38h (Enable external IVFC level 4).
The layout type 0 (external IVFC level 4 disabled) contains

```
DPFS level 1
DPFS level 2
DPFS level 3, and inside
IVFC level 1
IVFC level 2
IVFC level 3
IVFC level 4 (the actual content data)
```

The layout type 1 (external IVFC level 4 enabled) contains

```
DPFS level 1
DPFS level 2
DPFS level 3, and inside
IVFC level 1
IVFC level 2
IVFC level 3
IVFC level 4 ;the actual content data, note that this is outside DPFS level 3
```

DPFS tree
Everything inside the DPFS tree comes in pairs, and at one time only one of a pair is active. The tree is probably designed for atomic writing: for a file writing operation, it writes to the inactive part, then commits the data by switching a bit to activate it.

Each level of DPFS tree consists of a pair of chunks. The size of one chunk is defined as it in the DPFS descriptor, so the total size of a level is actually twice as large as the size recorded in the descriptor. For level 1 and 2, each chunk is a bit array, in which each bit corresponds to a block in the next level (the block size of the next level is also defined in the DPFS descriptor). This bit indicates which one of the pair in the next level is active for this block: 0 means the first one and 1 means the second one. The active chunk of level 1 is selected

by DPFS tree level 1 selector in the DIFI header. The bit array is encoded in u32 array, with MSB as the first bit of each 32 bits.

To access data in level 3, one needs to check the bits in level 1 and level 2 to know which chunk of level 3 is active for the accessed location. For example, for a following configuration:

```
  Level 1: size = 4 bytes = 32 bits
  Level 2: size = 0x380 bytes = 0x1C00 bits, block size = 0x80 bytes
  Level 3: size = 0x1B7F000, block size = 0x1000, block size = 0x1000 bytes
```

if one want to read byte at 0x1234567 of level 3, the following calculation is performed:

```
  get level 2 bit index 0x1234567 / 0x1000 = 0x1234,
    and its byte location 0x1234 / 8 = 0x246
  get level 1 bit index 0x246 / 0x80 = 4
  get level1_selector from DIFI header
  read level2_selector = Level1[level1_selector].bits[4];
  read level3_selector = Level2[level2_selector].bits[0x1234];
  read data = Level3[level3_selector].bytes[0x1234567] as the final data
```

in the code above Levelx[k] means the k-th chunk in level x, where k = 0, 1. .bits[n] is expanded to (.u32_array[n / 32] >> (31 - n % 32)) & 1 as the bit array is encoded in u32 array.
Effectively, the active data is scattered among the two level 3 chunk. One can assemble the whole active level 3 image following the same rule.

IVFC tree
The IVFC tree is used for data verification. It is very similar to the IVFC tree in RomFS, except that it has an additional level here. For level 1, 2 and 3, each level is a list of SHA-256 hash, of which each corresponds to a block of the next level which is zero-padded to align the block size (the block size of the next level is defined in the IVFC descriptor).

The partition master hash in the partition descriptor can be seen as IVFC level 0, which hashes level 1 following the same rule. The master hash is usually 20h long, consisting only one hash. This is because most DISA / DIFF files are not large enough to have multiple hashes on the top level, which isn't the case for some title database files.

However, not all data is hashed - only ranges that have been written with valid data are properly hashed.

Level 4 is the actual content of the partition, which is what the container format essentially contains.

Extracting content from a DISA / DIFF container
Find the active partition table and the partition(s).
Unwrap DPFS tree of partition(s) by reconstructing active data.
Unwrap IVFC tree. Either take out level 4 directly, or, better, verify all the hashes and poison the data that is not properly hashed.
The IVFC level 4 is the inner content of the file. The format of it varies among different usage. Refer their own page for further extraction.

Chain of trust
AES CMAC verifies the header.
The header verifies the active partition table via the table hash.
In the partition table, each descriptor verifies level 1 of its IVFC tree via the master hash.
Each IVFC level verifies the next level, until the level 4, which is the inner content.

Summary diagram
external link: https://i.imgur.com/BjwShJZ.png
Please move this into 3dbrew when file uploading is fixed.

3DS Icon SMDH

This page describes the format of the icon stored at CXI ExeFS:/icon and CIA icons. The CXI icon is displayed by Home Menu and System Settings (3DS Software Management), while CIA icons are dummies and not yet utilised by Dev 3DS' (as of rev 47586).

Icon File Format (36C0h bytes)

```
  0000h 4     ID "SMDH"
  0004h 2     Version (unknown/unspecified)
  0006h 2     Reserved
  0008h 200h  Title Japanese             ;\
  0208h 200h  Title English              ;
  0408h 200h  Title French               ; Each 200h-byte entry consists of:
  0608h 200h  Title German               ;   80h-byte   Short Description
  0808h 200h  Title Italian              ;   100h-byte  Long Description
  0A08h 200h  Title Spanish              ;   80h-byte   Publisher
  0C08h 200h  Title Simplified Chinese   ; All encoded in UTF-16
  0E08h 200h  Title Korean               ;
  1008h 200h  Title Dutch                ;
  1208h 200h  Title Portuguese           ;
  1408h 200h  Title Russian              ;
  1608h 200h  Title Traditional Chinese  ;
  1808h 200h  Title Reserved             ;
  1A08h 200h  Title Reserved             ;
  1C08h 200h  Title Reserved             ;
  1E08h 200h  Title Reserved             ;/
  2008h 1     Age Rating CERO (Japan)
  2009h 1     Age Rating ESRB (USA)
  200Ah 1     Age Rating Reserved
  200Bh 1     Age Rating USK (German)
  200Ch 1     Age Rating PEGI GEN (Europe)
  200Dh 1     Age Rating Reserved
  200Eh 1     Age Rating PEGI PRT (Portugal)
  200Fh 1     Age Rating PEGI BBFC (England)
  2010h 1     Age Rating COB (Australia)
  2011h 1     Age Rating GRB (South Korea)
  2012h 1     Age Rating CGSRR (Taiwan)
  2013h 1     Age Rating Reserved
  2014h 1     Age Rating Reserved
  2015h 1     Age Rating Reserved
  2016h 1     Age Rating Reserved
  2017h 1     Age Rating Reserved
  2018h 4     Region Lockout (bit0=JPN, bit1=USA, bit2=EUR, bit3=AUS, bit4=CHN,
                bit5=KOR, bit6=TWN, bit7-31=Reserved) (7FFFFFFFh=Region Free)
  201Ch 4     Match Maker ID      ;\Match Maker IDs (Online Play)
  2020h 8     Match Maker BIT ID  ;/
  2028h 4     Flags
  202Ch 1     EULA Version Minor  ;\
  202Dh 1     EULA Version Major  ;/
  202Eh 2     Reserved
  2030h 4     'Optimal Animation Default Frame' (for BNR)
  2034h 4     CEC (StreetPass) ID (So the Home Menu knows which application
                icon to show the 'Green' CEC notification for)
  2038h 8     Reserved
  2040h 480h  Small Icon (24x24pix, shown on top screen when pausing the app)
  24C0h 1200h Large Icon (48x48pix, the general icon)
```

Flags [2028h]

```
  0     Visibility Flag (Required for visibility on the Home Menu)
  1     Auto-boot this gamecard title
  2     Allow use of 3D? (For use with parental Controls. An application
          can use the 3D affect, even when this flag isn't set)
```

```
 3     Require accepting CTR EULA before being launched by Home
 4     Autosave on exit? (see below)
 5     Uses an Extended Banner?
 6     Region game rating required
 7     Uses save data? (see below)
 8     Application usage is to be recorded. If this is zero, it causes
         the application's usage to be omitted from the Home Menu's icon
         cache, as well as in other places.
 9     unknown/unspecified
 10    Disables SD Savedata Backups for this title. This is in addition to
        the blacklist.
 11    unknown/unspecified
 12    New 3DS exclusive title. Shows an error if used on Old 3DS.
 13-31 unknown/unspecified
```

Age Rating: Active ratings have a bitmask of 80h, and inactive ratings have no bitmask at all. Ratings without the 80h bitmask are ignored. 40h bitmask indicates Rating Pending. 20h bitmask indicates No Age Restriction. Age limits are set by adding the minimal age to 80h (for example, limiting to 12 years and up would give a bitmask of 8Ch)

Region Lockout: Regions are 'included' in region lock by setting their bitmask value. Nintendo defines region free as 7FFFFFFFh. Early in the 3DS' development, Nintendo grouped the Australian and Europe markets together. Nintendo defines market Europe as having the combined bitmasks of Europe and Australia. No 3DS' which check the Australia bitmask have been seen (Australia uses the European 3DS model).

The EULA version is checked when the Accept EULA flag is set, the version is compared to one stored in the 3DS. If the SMDH version is greater, then the user will be prompted to accept the EULA.

Effect of SaveData and AutoSave
These options have no effect on the performance of the application itself: they're used to select an appropriate warning when closing an application from Home.
Both off: "Closing software" (no warning if quitting directly with X)
SaveData: "Do you want to close [...]? (Unsaved data will be lost.)"
AutoSave: ?
Both on: "Saving data and closing software..." (no warning if quitting directly with X)

'Optimal Animation Default Frame' (for BNR)
This is a float, indicating the preferred (or 'most representative') frame for the banner animation.

CEC (StreetPass) ID
This u32 represents the application CEC ID. This is likely loaded by applications for use with the CEC services as well.

Icon graphics
Both of the icons are encoded in RGB565 meaning 16bpp.
The data is encoded in tiles (starting from size 8x8, continuing recursively).

```
If the buffer is like this:
 0       1       2       3       4       5       6       7       8       9
 10      11      12      13      14      15      16
Then the image would look like this:
 x=0     x=1     x=2     x=3     x=4
 0       1       4       5       16
 2       3       6       7       ...
 8       9       12      13
 10      11      14      15
Uh, are that pixels or 8x8pix tiles?
Either way, how would that translate to actual 24x24pix or 48x48pix icons??
```

Although both icons are known to be RGB565, developers have the option of encoding icons (and banners) with the following encodings:

```
RGBA8
RGB8
RGBA5551
RGB565     (used)
RGBA4
LA8
HILO8
L8
A8
LA4
L4
ETC1
ETC1A4
```

This does not necessarily mean the other encodings will be used, it is just that those are the options when compiling. Like we've seen with Super Mario 3D Land Nintendo has changed save file encryption, and "likewise" they can encode icons and banners differently should they choose to. Currently we've seen just RGB565 so don't be fooled if an icon doesn't show up right! It is probably one of these formats above. Although we will probably not see other formats used for a while it's nice to know they have an opportunity to change. Also note that it seems Nintendo stores "each pixel in word-order", so the actual "order of order" of "each color channel" in memory will depend on the endianness. uh, what?

Tools

```
CiTRUS - (GUI)(Windows Only) Generating ICN files
3DSExplorer - (GUI)(Windows Only) Parsing ICN files
```

3DS Banner CBMD Header

CBMD - CTR Banner Model Data
This page describes the format used for banners' models. These are stored in CXI ExeFS:/banner and optionally in extdata exbanner. CBMD is a container file for CGFX blocks. This is used for banners of titles you see in the home menu. BNR used for the app banners in the CXI/exbanner is the same as CBMD with CWAV at the end (uh, who is "BNR", and is "CWAV" same as "BCWAV", and is that (B)CWAV always supported in all cases, or only for that "BNR" thing?).

CBMD Header

```
000h 4     Magic "CBMD"
004h 4     Zero
008h 4     CGFX Offset for default        ;-default CGFX
00Ch 4     CGFX Offset for EUR-English    ;\
010h 4     CGFX Offset for EUR-French     ;
014h 4     CGFX Offset for EUR-German     ; optional region-specific
018h 4     CGFX Offset for EUR-Italian    ; CGFX (or 0=use default CGFX)
01Ch 4     CGFX Offset for EUR-Spanish    ;
020h 4     CGFX Offset for EUR-Dutch      ;  (unknown if CHN/KOR/TWN are
024h 4     CGFX Offset for EUR-Portuguese ;  also supported)
028h 4     CGFX Offset for EUR-Russian    ;
02Ch 4     CGFX Offset for JPN-Japanese   ;
030h 4     CGFX Offset for USA-English    ;
034h 4     CGFX Offset for USA-French     ;
038h 4     CGFX Offset for USA-Spanish    ;
03Ch 4     CGFX Offset for USA-Portuguese ;/
040h 44h   Padding?
084h 4     BCWAV Offset
```

The common CGFX is used if the CGFX offset for the system region/language is zero. Those optional offsets can be used in extdata exbanner, but separate CBMD banner files for each region/language can be used as well.

CGFX
CGFX are compressed using LZ11. For CXI banner CGFX, the decompressed size must be no larger than 80000h.
Graphics containers. Contains: 3D Models, Shaders, Objects, Materials, Textures, etc. See CGFX for more information.

3DS Banner CGFX Graphics

CGFX is a container format used to store graphics resources. It can contain 3D models, textures and animation data.

CGFX header

```
  000h 4     Magic "CGFX"
  004h 2     Byte order mark (FFFEh=little endian, FEFFh=big endian) (uh, ehm?)
  006h 2     CGFX header size
  008h 4     Revision
  00Ch 4     File size (bytes)
  010h 4     Number of entries
```

A typical CGFX file contains two main entries, beginning directly after the CGFX header: DATA and IMAG (uh, what is "IMAG"?).

DATA
DATA contains a list of DICT references.
DATA header (for N = 0..15):

```
  000h       4     Magic "DATA"
  004h       4     DATA Size (in bytes)
  008h+(N*8) 4     Number of entries in DICT N
  00Ch+(N*8) 4     Offset (self-relative) to DICT N
```

The DATA header contains the entry counts and offsets for each DICT entry. The number of entries can vary (probably based on the version?), but are always in the following order. Any unused entries are zeroed.

```
 Typical entries:
  N       Type
  0       Models
  1       Textures
  2       LUTS (Material/Color/Shader look-up tables?)
  3       Materials
  4       Shaders
  5       Cameras
  6       Lights
  7       Fog
  8       Environments
  9       Skeleton animations
  10      Texture animations
  11      Visibility animations
  12      Camera animations
  13      Light animations
  14      Emitters
  15      Unknown
```

DICT
DICTs are generic structures used to store values (and associate them to a key ?). A DICT header is 1Ch bytes long.

```
 DICT header:
  00h 4     Magic "DICT"
  04h 4     DICT size (in bytes)
  08h 4     Number of entries
  0Ch 4     ?
  10h 2     Unknown. Seems to be shifted left by 4 bits in the source.
```

```
  10h 2     ?
  12h 0Ah   ?
 DICT entry:
  00h 4     ?
  04h 2     ?
  06h 2     ?
  08h 4     Offset (self-relative) to symbol
  0Ch 4     Offset (self-relative) to object
```

CMDL

CMDL is used to describe a 3D model.

```
 CMDL Header:
  00h 4     Flags (bit 7: hasSkeletonSobj)
  04h 4     Magic "CMDL"
  08h 4     ?
  0Ch 4     Offset (self-relative) to model name
  10h 18h   ?
  28h 4     Number of entries in Animation Types DICT
  2Ch 4     Offset (self-relative) to Animation Types DICT
  30h 0Ch   Global scale vector (3 floats: x, y, z)
  3Ch 18h   ?
  54h 30h   Matrix 1
  84h 30h   Matrix 2
  B4h 4     Number of Vertex Info SOBJ entries
  B8h 4     Offset (self-relative) to Vertex Info SOBJ list
  BCh 4     Number of MTOB DICT entries
  C0h 4     Offset (self-relative) to MTOB DICT
  C4h 4     Number of Vertex Info SOBJ entries
  C8h 4     Offset (self-relative) to Vertex Info SOBJ list
  CCh 4     Number of Unknown DICT entries
  D0h 4     Offset (self-relative) to Unknown DICT
  D4h 0Ch   ?
  E0h 4     Skeleton Info SOBJ offset (self-relative) (present if flag bit7=1)
  [B8h]+B8h 4*N   Vertex Info SOBJ self-relative offset list
```

A CMDL section refers to outside data; it can not be considered separately from the rest of the CGFX file. The second DICT in the CMDL section contains offsets to MTOB objects.

SOBJ

SOBJ structures can be used to describe 3D objects that are part of the model. If such is the case then they will follow this structure:

```
  00h 4     Flags (bit 4: model; bit 1: skeleton)
  04h 4     Magic "SOBJ"
  08h 4     ?
  0Ch 4     Unknown symbol offset (self-relative)
  10h 0Ch   ?
  1Ch 4     Offset (self-relative) to Unknown1 (array of floats?)
  20h 0Ch   Mesh position offset (X/Y/Z floats)
  2Ch 4     Face groups count
  30h 4     Offset (self-relative) to face groups offset array
  34h 4     ?
  38h 4     Vertex groups count
  3Ch 4     Offset (self-relative) to vertex groups offset array
  40h 4     Unknown offset (self-relative) ?
```

Face groups:

```
  00h 4     Bone groups count
  04h 4     Offset (self-relative) to UInt32 bone group IDs array
  08h 4     ?
  0Ch 4     Unknown2 count
  10h 4     Offset (self-relative) to Unknown2 offset array
```

Unknown2:

```
  00h 4     Face group descriptor count
  04h 4     Offset (self-relative) to face array descriptors offset array
  08h 4     Unknown3 count
  0Ch 4     Offset (self-relative) to UInt32 Unknown3 array
  10h 8     ?
```

Face array descriptor:

```
  00h 4     Flags (bit 1: vertex index format: 0=byte, 1=short)
  04h 4     ?
  08h 4     Vertex index array size (in bytes)
  0Ch 4     Offset (self-relative) to vertex index array
```

Vertex groups come in a number of different formats. Typically the first vertex group entry is of format 40000002h and contains the actual vertex array.

Vertex group format 40000002h:

```
  00h 4     Flags (40000002h)
  04h 4     ?
  08h 4     ?
  0Ch 4     ?
  10h 4     ?
  14h 4     Vertex array size (in bytes)
  18h 4     Offset (self-relative) to vertex array
  1Ch 4     ?
  20h 4     ?
  24h 4     Vertex stride/size in bytes (see below)
  28h 4     Unknown3 count
  2Ch 4     Offset (self-relative) to component declaration offset array
```

Each mesh's primary vertex group contains an array of vertex component declaration objects, defining the order and parameters for each of a vertex's components.

Vertex component declaration:

```
  00h 4     Flags (40000001h)
  04h 4     Vertex component type (see below)
  08h 4     ?
  0Ch 4     ?
  10h 4     ?
  14h 4     ?
  18h 4     ?
  1Ch 4     ?
  20h 4     ?
  24h 1     Vertex component data type (see below)
  25h 1     ?
  26h 1     ?
  27h 1     ?
  28h 4     Number of values in this component (e.g. XYZ->3, UV->2)
  2Ch 4     Multiplier for this component's values (float)
  30h 4     Position of this component within vertex stride
```

Vertex formats with bone data support multiple bone assignment. In this case, the sum of all bone weights is 64h.

```
 Vertex component types:
  Value   Type
  00h = Position
  01h = Normal
  02h = ? (unobserved)
  03h = Color
  04h = UV0
  05h = UV1
  06h = ? (unobserved, possibly UV2)
  07h = Weight
  08h = Index
```

```
 Vertex component data types:
  Value   Type
  00h = sbyte
  01h = byte
  02h = short
  03h = ? (unobserved, possibly ushort)
  04h = ? (unobserved, possibly int)
  05h = ? (unobserved, possibly uint)
  06h = float
```
Vertex components are stored as one of the above data types, and the vertex component declaration contains a multiplier that adapts the values to the float version which the game will use. For example, color RGBA values are stored as bytes, and the multiplier converts them from 0-255 to 0-1.0, and position components using short values are normalized via the multiplier to take advantage of the entire short value range.

TXOB

TXOBs are contained within MTOBs. They can describe textures; if such is the case, then their structure is as follows:
```
  00h 4     Flags
  04h 4     Magic "TXOB"
  08h 8     ?
  0Ch 4     Offset (self-relative) to symbol
  18h 4     Texture height
  1Ch 4     Texture width
  28h 4     Mipmap levels
  34h 4     Texture format ID (see table below)
  3Ch 4     Texture height (?)
  40h 4     Texture width (?)
  44h 4     Texture data size
  48h 4     Texture data offset (self-relative)
```
Texture format:
```
  00h = RGBA8
  01h = RGB8
  02h = RGBA5551
  03h = RGB565
  04h = RGBA4
  05h = LA8
  06h = HILO8
  07h = L8
  08h = A8
  09h = LA4
  0Ah = L4
  0Bh = A4 ?
  0Ch = ETC1 (see notes below)
  0Dh = ETC1A4 ?
```
Every texture format has its texture data divided into 8x8 tiles. See SMDH for more information. ETC1 is a compressed texture format which compresses blocks of 4x4 pixels into u64s. These u64 are traditionally stored in big endian; however, nintendo's implementation stores them in little endian. ETC1 textures are stored in 8x8 tiles; decompressed 4x4 therefore have to be organized accordingly. See [1] for implementation example.

LUTS

Appears to contain color lookup tables possibly for use with shaders.

LUTS Header:
```
  00h 4     Magic "LUTS"
  04h 2     Seems to adhere to powers of 2 (width/height/flags?)
  06h 2     Seems to adhere to powers of 2 (width/height/flags?)
  08h 4     ?
  0Ch 8     all zeroes ?
  14h 4     ?
  18h 4     Offset to DICT (self-relative) ?   ;this is "4" (?)
```
All observed instances have an otherwise unreferenced DICT section immediately afterward (the last LUTS value being a "4", which may describe the relative position of that DICT), which appears to describe material

specularity.

Skeleton data
Each entry is E0h bytes in length and organized this way:

```
00h 4     Offset (self relative) to name symbol
04h 4     ?
08h 4     Joint ID
0Ch 4     Parent joint ID
10h 4     Signed offset (self-relative) to parent joint
14h 18h   unknown/unspecified
2Ch 0Ch   Angle vector (floats, x, y, z)
38h 0Ch   Position vector (floats, x, y, z)
44h 30h   Transformation matrix (4x3)
74h 30h   Identity matrix ? (4x3)
A4h 1Ch   unknown/unspecified
```

Each entry stores the joint transformation data twice; once as angle/position vectors and once as a transformation matrix. Each entry also stores a second matrix which appears to always be identity(?)

CANM
CANMs are used to store skeletal animation data.

Tools

```
Every File Explorer
Ohana3DS and its forks
SPICA
```

Links
Another CGFX Format Description (Archived Page):
http://florian.nouwt.com/wiki/index.php/CGFX_(File_Format)

3DS Banner CWAV Sound

This document is about the format of Banner's CTR Wave files (BCWAV).

The structure is very similar to Microsoft's Wave file.

Banner CWAV total channels must be 2, and the length of the audio must be 3 seconds or less, otherwise the sound will play incorrectly (beeping/clicking) or the model may fail to load.

Overview
Microsoft's WAV structure is RIFF Header which defines the data inside which is WAVE, then the media player expects a "fmt " chunk and a "data" chunk. Nintendo's format uses a CWAV header (no need for a general structure for media, only wave), which points to an INFO struct (the equivalent to fmt) and a DATA struct (the equivalent to data).

Header

```
000h 4   Magic (CWAV)
004h 2   Endianness (FEFFh=little, FFFEh=big) (uh, ehm, in which endian?)
006h 2   Header Size (40h due to Info Block alignment)
008h 4   Version (02010000h)
00Ch 4   File Size
010h 2   Number of Blocks (2)
012h 2   Reserved
014h 12  Info Block Sized Reference (Offset relative to start of file)
020h 12  Data Block Sized Reference (Offset relative to start of file)
```

Info Block

```
  000h 4   Chunk ID "INFO"
  004h 4   Chunk Size
  008h 1   Encoding (0=PCM8, 1=PCM16, 2=DSP-ADPCM, 3=IMA-ADPCM)
  009h 1   Loop (0=Don't loop, 1=Loop)
  00Ah 2   Padding
  00Ch 4   Sample Rate
  010h 4   Loop Start Frame
  014h 4   Loop End Frame
  018h 4   Reserved
  01Ch 8   Samples Reference    (Offset relative to Data Block Data field)
  024h 8   ADPCM Info Reference (Offset relative to Samples Reference field)
  02Ch 4   Reserved
 If encoding is DSP ADPCM:
  X    X   DSP ADPCM Info Entries
 If encoding is IMA ADPCM:
  X    X   IMA ADPCM Info Entries
```

The info block is aligned to 20h bytes.

DSP ADPCM Info

```
  000h 32  16-bit Coefficients
  020h 1   4-bit Predictor + 4-bit Scale ;\
  021h 1   Reserved                      ; Initial context
  022h 2   Previous Sample               ;
  024h 2   Second Previous Sample        ;/
  026h 1   4-bit Predictor + 4-bit Scale ;\
  027h 1   Reserved                      ; Loop context
  028h 2   Previous Sample               ;
  02Ah 2   Second Previous Sample        ;/
  02Ch 2   Padding
```

IMA ADPCM Info

```
  000h 2   Data                          ;\
  002h 1   Table Index                   ; Initial context
  003h 1   Padding                       ;/
  004h 2   Data                          ;\
  006h 1   Table Index                   ; Loop context
  007h 1   Padding                       ;/
```

Data Block

```
  000h 4   Chunk ID "DATA"
  004h 4   Chunk Size
  008h ..  Data (chunk size minus 8 bytes)
```

The data block is aligned to 20h bytes, as well as the data field's actual sample data.

Reference Table - uh, what/where is that?

```
  000h 4   Count (N)
  004h N*8 References (Offsets relative to Count field)
```

Reference

```
  000h 2   Type ID
  002h 2   Padding
  004h 4   Offset (FFFFFFFFh="null")   ;uh, maybe "null" means none?
```

Sized Reference

```
  000h 8   Reference
  008h 4   Size
```

Reference Type IDs

```
0300h = DSP ADPCM Info
0301h = IMA ADPCM Info
1F00h = Sample Data
7000h = Info Block
7001h = Data Block
7100h = Channel Info
```

3DS Banner Extended Banner

Overview
The Extended Banner is used to add text (and optionally an extra texture) to a given banner. It can also (optionally) be used to set an expiration date for a banner.

See here (uh, where?) for how the extended-banners are loaded from extdata. The Home Menu extended-banner loading function will immediately return without loading anything if the programID is for System Settings.

Format
Extension: .BIN

```
000h 2    Texture width (if texture is used)
002h 2    Texture height (if texture is used)
004h 2    Texture colour format (if texture is used)
008h 4    Year to expire
00Ch 2    Month to expire
010h 2    Day to expire
014h 15h  Name of texture (if texture is used) ;uh, 15h maybe is 16 decimal?
024h 200h Plain text comment (255 character max) to be displayed in the
            banner on the HOME Menu
224h 80h  unknown/unspecified
2A4h ..   If used, this is where the extra texture is located, otherwise
            file ends.
```

The date used for never-expiring exbanners varies, on retail like with Mii Maker this is 31 Dec 2099 (year=833h, month=0Ch, day=1Fh). The expiration-timestamp is only used for SpotPass exbanners, not extdata-exbanners.

Texture Colour Formats

```
00h = RGBA8
01h = RGB8
02h = RGBA5551
03h = RGBA565
04h = RGBA4
05h = LA8
06h = HILO8
07h = L8
08h = A8
09h = LA4
0Ah = L4
0Bh = A4
0Ch = ETC1
0Dh = ETC1A4
```

SpotPass
When Home Menu loads extended-banners, it also attempts to load a "CBMD" banner via SpotPass service commands. Normally this CBMD banner doesn't exist in extended-banner extdata. This is broken with New3DS titles since Home Menu uses these BOSS commands with the New3DS bitmask in the programID set. The common and language-specific (when offset is non-zero) "CGFX" specified by the CBMD are decompressed and processed.

The "CGFX" sections in this CBMD are actually the exact same exbanners loaded from normal extdata. The

exbanner data from SpotPass is stored to the same state as the extdata-exbanners. No CWAV is loaded from SpotPass data.

The exbanners from SpotPass must have a timestamp less than current_datetime, otherwise they won't be parsed. The timestamp for the banner is calculated with: nintimestamp_mktime(out, exbanner->year, exbanner->month, exbanner->day, <hour=23>, <minute = 59>, <second = 59>, <millisecond = 999>);

Titles using extended banners
System:
```
Mii Maker
Face Raiders
```

3DS Module NWM (Wifi Driver)

NWM File/versions
The wifi driver can be found in following locations (each console should have one normal mode file, and one safe mode file):
```
3ds:\title\00040130\00002d02\content\000000vv.app  ;NWM Normal mode (all 3DS)
3ds:\title\00040130\00002d03\content\0000000v.app  ;NWM Safe mode (Old3DS)
3ds:\title\00040130\20002d03\content\0000000v.app  ;NWM Safe mode (New3DS)
```
The .app file contains a compressed .code file (inside of the NCCH ExeFS). That .code file contains ARM11 code and six datablocks with the actual Xtensa wifi firmware (similar as in the DSi wifi firmware):
```
Stub.data   038h bytes                  ;\Database and EEPROM reading stub
Stub.code   316h bytes                  ; (always same size/content)
Database    1E8h bytes                  ;/
Main.type1  0FD3h, 10F7h, or 1B1Bh bytes ;\Main firmware (compressed)
Main.type4  A053h, A482h, or A5EBh bytes ; (size/content varies per version)
Main.type5  78F6h, or 7A2Eh bytes       ;/
```
There are a few safe mode versions, and dozen(s) of normal mode versions. The versions may contain different ARM11 code, and/or different Xtensa code (as shown above, there are at least three revisions of the Xtensa "Main" code blocks (spotted in four different NWM files, so there may be much more variations)).

NWM Main Type1/4/5
There are three variants of the Main firmware, called Type1/4/5 (because the ARM11 code uses "CMP r0,1/4/5" opcodes to determine which one to use).
```
Type1 = Standard functions for normal internet access
 Type1 supports the same commands/events as on DSi
Type4 = Special AP Mode and whatever:
 WMIcmd(004Ah..0052h) seem to AP mode (alike Atheros cmd F00Bh..F013h)
 WMIcmd(0053h,0055h..005Fh) unknown
 WMIevent(101Dh,101Fh,1020h,1022h,1024h,1025h) unknown
 Type4 can be used for normal internet access (when ignoring event 1025h)
Type5 = Special MacFilter, GameID, and built-in SHA1 function
 Contains ascii strings "MACFilter" and "GameID", and a built-in SHA1 function
 WMIcmd(0060h..0073h) unknown
 WMIevent(1021h,1023h,1026h,1027h) unknown
 Type5 doesn't seem to support normal internet access (it can't even find APs)
```
Type4/5 are probably for Nintendo-specific special local network multiplayer features, possibly related to "Streetpass" (in sleep mode?) and "DownloadPlay" (for single gamepak), and/or other purposes.

Addresses in ARM11 Pool
The ARM11 code does reference the Xtensa datablocks with "LDR Rd,=address" opcodes, hence having the datablock addresses stored in the literal pool. The pool isn't exactly intended to be used as "table of contents", but it can be misused for that purpose:
Search for constant 00524C00h, check that constant 000003EDh is located at the expected offset, then extract the datablock start/end addresses from the corresponding pool locations, subtract 10000h to convert addresses to

.code file offsets.
The NWM file does contain two pools: One pool with the Main type1/4/5 addresses, followed by by larger pool with the same addresses, plus addresses for the Database and EEPROM reading stub code/data.
Most files (3 of 4 known files) are using this pool structure:

```
Newer Small Pool:
00h main.dst (and stub.data.dst) ;CONST (00524C00h)
04h whatever value               ;CONST (000003EDh)
08h main.src.type1.end    ;\Type1 (basic internet)
0Ch main.src.type1.start  ;/
10h main.src.type4.end    ;\Type4 (ApMode and whatever)
14h main.src.type4.start  ;/
18h main.src.type5.end    ;\Type5 (MacFilter,GameId,Sha1)
1Ch main.src.type5.start  ;/
Newer Large Pool:
00h whatever.addr.plus00h         ;\
04h whatever.addr.plus04h         ; INCREASING addresses
08h whatever.addr.plus08h         ;
0Ch whatever.addr.plus0Ch         ;/
10h whatever.other.addr
14h main.dst (and stub.data.dst) ;CONST (00524C00h)
18h whatever value               ;CONST (000003EDh)
1Ch main.src.type1.end   ;\Type1 (basic internet)
20h main.src.type1.start ;/
24h main.src.type4.end   ;\Type4 (ApMode and whatever)
28h main.src.type4.start ;/
2Ch main.src.type5.end   ;\Type5 (MacFilter,GameId,Sha1)
30h main.src.type5.start ;/
34h database.src.end     ;\Datab ;SAME as stub.code.src.start
38h database.src.start   ;/
3Ch database.dst                 ;CONST (0053FE18h)
40h stub.code.src.end    ;\Stubc
44h stub.code.src.start  ;/      ;SAME as database.src.end
48h stub.code.dst                ;CONST (00527000h)
4Ch stub.data.src.end    ;\Stubd
50h stub.data.src.start  ;/
54h whatever.thumb.code.addr1
58h whatever.thumb.code.addr2
5Ch whatever.thumb.code.addr3
```

There is at least one older file using this pool structure:

```
Older Small Pool:
00h main.dst (and stub.data.dst) ;CONST (00524C00h)
04h maintype5.src.end
08h maintype5.src.start
0Ch maintype1.src.end
10h maintype1.src.start
14h maintype4.src.end
18h maintype4.src.start
1Ch whatever value               ;-CONST (000003EDh)
Older Large Pool:
00h whatever.addr.plus00h        ;\
04h whatever.addr.plus04h        ;
08h whatever.addr.plus08h        ; INCREASING addresses
0Ch whatever.addr.plus0Ch        ;
10h whatever.addr.plus10h        ;/
14h whatever.other.addr
18h main.dst (and stub.data.dst) ;CONST (00524C00h)
1Ch maintype5.src.end
20h maintype5.src.start
24h database.dst                 ;CONST (53FE18h)
28h database.src.end             ;SAME as stub.code.src.start
2Ch database.src.start
30h stub.code.dst                ;CONST (527000h)
34h stub.code.src.end
38h stub.code.src.start          ;SAME as database.src.end
```

```
  3Ch stub.data.src.end
  40h stub.data.src.start
  44h whatever.thumb.code.addr1
  48h whatever.thumb.code.addr2
  4Ch whatever.thumb.code.addr3
  50h maintype1.src.end
  54h maintype1.src.start
  58h maintype4.src.end
  5Ch maintype4.src.start
  60h whatever value                ;CONST (000003EDh)
```
Moreover, there are dozens of unknown files, which may (or may not) use yet different pool structures.

3DS Console IDs

Hardware IDs
The main Console ID is stored in OTP memory:
[3DS Crypto - PRNG and OTP Registers]
The eMMC CID is used for generating console specific encryption keys. The SD card CID is also used for generating a SD card specific <ID1> folder name.

3ds:\rw\sys\LocalFriendCodeSeed_B (or Seed_A, if it exists) (110h bytes)
```
  000h 100h  RSA-2048 signature across following 10h-bytes  ;\
  100h 1     Zero                                           ; same values are
  101h 1     Devkit (00h=Retail, 01h=Devkit)                ; also stored in
  102h 6     Zero                                           ; movable.sed
  108h 8     Decrypted OTP[08h..0Fh]                        ;/
```
This can be also retrieved via "PSPXI:GetLocalFriendCodeSeed".

3ds:\private\movable.sed (120h or 140h bytes)
```
  000h 4     ID "SEED"
  004h 1     Zero
  005h 1     Zero (or 01h if extra bytes at [120h..13Fh] are appended)
  006h 2     Zero
  008h 100h  RSA-2048 signature across following 10h-bytes  ;\same values are
  108h 1     Zero                                           ; also stored in
  109h 1     Devkit (00h=Retail, 01h=Devkit)                ; LocalFriendCode
  10Ah 6     Zero                                           ; Seed_B (or _A)
  110h 8     Decrypted OTP[08h..0Fh]        ;\used as AES   ;/
  118h 4     Decrypted OTP[10h..13h]+Offset ; KeyY for 3
  11Ch 4     Decrypted OTP[14h..17h]        ;/keyslots
 The original movable.sed from the factory is only 120h-bytes.
 Below extra data is written to the file when doing a System Format.
  120h 4     Offset (added to above Decrypted OTP[10h..13h] entry)
  124h 0Ch   Zero
  130h 10h   AES-MAC (NAND dbs keyslot) across SHA256 across bytes [000h..12Fh]
```
Bytes [110h..11Fh] are used as KeyY for AES keyslots 30h, 34h, and 3Ah:
```
  Keyslot 30h  AES-CMACs for non-DSiWare in sd:\Nintendo 3DS\ and 3ds:\data\
  Keyslot 34h  AES encryption in sd:\Nintendo 3DS\
  Keyslot 3Ah  AES-CMACs for DSiWare in sd:\Nintendo 3DS\
```
Bytes [110h..11Fh] are also used to derive the <ID0> folder name (see below).
Movable.sed is transferred to the destination 3DS during a System Transfer. The movable.sed keyY high u64 is updated on the source 3DS during a System Transfer, and when doing a system format with System Settings.

3ds:\data\<ID0>\ (<ID0> folder on 3DS partition)
sd:\Nintendo 3DS\<ID0>\<ID1>\ (<ID0>\<ID1> folder on SD card)
The <ID0> folder name is derived by computing the SHA256 across moveable.sed bytes [110h..11Fh], and then converting the first 4 words of the SHA256 into a lowercase ASCII string (with the byte order reversed in each of the four words, and with the last 4 words left unused).

The <ID1> folder name is derived by swapping around the byte order of the SD card CID: The CID is rotated 8-bits to the left (uh, that means the ending 00h padding byte is moved to begin?). Then, the word-order is reversed. And then, the halfwords are converted into ascii string (with byte-order for each byte-pair reversed), or so?

3ds:\rw\sys\SecureInfo_A (111h bytes) (Region and Serial/Barcode)

```
000h 100h  RSA-2048 signature across following 11h-bytes
100h 1     Region (0=JPN, 1=USA, 2=EUR, 3=Reserved, 4=CHN, 5=KOR, 6=TWN)
101h 1     Normally zero
102h 0Fh   Serial/Barcode, without ending check digit (ASCII, zeropadded)
```

The serial/barcode consists of 2-3 letters, followed by 8 digits, followed by a checksum digit.
The first letter indicates the console model:

```
3DS             C  (or E for devunits)
3DS XL/LL       S  (or R for devunits)
2DS             A  (or P for devunits)
New 3DS         Y  (or Yxx00 for devunits)
New 3DS XL/LL   Q  (or Qxx00 for devunits)
New 2DS XL/LL   N  (or Nxx01 for devunits)
```

The next 1-2 letter(s) indicate the region:

```
JPN Japan                          JF,JH,JM
USA North America                  W
USA Middle East, Southeast Asia    S
EUR Europe                         EF,EH,EM
EUR Australia                      AG,AH
CHN China (iQue)                   CF,CH,CM
KOR South Korea                    KF,KH,KM
TWN Taiwan                         ...unknown?
```

The checksum digit is found on the consoles barcode sticker, but isn't included in the SecureInfo_A file. However, the checksum can be calculated across 1st-8th digit (and ignoring the leading letters):

```
9th = (250 - (1st+3rd+5th+7th) - 3*(2nd+4th+6th+8th)) mod 10
```

The RSA private key for retail is unknown. The private key for devkits is included in the devkit system updater package.

3DS eMMC and MCU Images

eMMC Images

no$gba can load 3DS eMMC images from following files in no$gba folder (currently that merely allows to view eMMC images in Window/Filesystem, ie. there isn't any actual 3DS emulation supported in no$gba):

```
3DS-#.mmc      ;\with "#" being the machine number ("1".."C")
New3DS-#.mmc   ;/
```

For decrypting the image, CID and OTP needs to be stored at following eMMC offsets:

```
DC00h 20h   ID '3DS CID/OTP/NCSD/FIRM BackupData'
DC20h 10h   eMMC CID (from eMMC command CMD10)
DC30h D0h   Reserved
DD00h 100h  OTP   (from ARM9 I/O address 10012000h)
DE00h 200h  Copy of original NCSD  (from eMMC offset 00000000h)
E000h 1000h Copy of original FIRM0 (from eMMC offset 0B130000h)
F000h 1000h Copy of original FIRM1 (from eMMC offset 0B530000h)
```

The copy of original NCSD/FIRM0/FIRM1 allows to uninstall sighax-based stuff (or to install such stuff; doing that works even without having OTP dump, as long as the unencrypted content of the NCSD/FIRM0/FIRM1 sectors is known).

MCU Firmware Images

The MCU image can be viewed in no$gba's RL78 disassembler (via Alt+W, R). The image is loaded from following files in no$gba directory:

```
3DS.mcu
New3DS.mcu
```

The file should contain a copy of the whole 8000h-byte MCU flash memory:

```
  0000h 1000h Part 1 current version
  1000h 1000h Part 1 old version backup
  2000h 3000h Part 2 current version
  5000h 3000h Part 2 old version backup
```

If the old version areas aren't dumped, best leave them empty.

Atheros Wifi ROM image

The Wifi ROM image can be viewed in no$gba's Xtensa disassembler (via Alt+W, X). The image is loaded from following file (in no$gba directory):

```
  AR6014G.ROM  ;3DS wifi chip     (256Kbytes)
```

3DS Component Lists

3DS Component List

```
 PCB "C/CTR-CPU-01"
  U1   bga   "1048 0H, CPU CTR, (M)(C)2010, Nintendo, JAPAN ARM" (main cpu)
  U2   bga   "F JAPAN, MB82M8080-07L, 1040 M90, E1"              (main ram)
  U3   bga   "Texas Instruments, 93045A4, 0AAH86W GI" (PCB back, below YXAB)
  U4   bga   "Texas Instruments, PAIC301DB, 0AA37DW, GI"         (TSC)
  U5   bga   "TOSHIBA THGBM2G3P1FBAI8, VX2306, TAIWAN, 10459AE"  (eMMC)
  U6   bga   "UC CTR, 041KM73, KG10"                             (MCU)
  U?? 16pin  "CKP, TI 09W, ZF1T"     (PCB back, above YXAB)  (battery charger)
  U8   8pin  "17040, 08A45" MAX17040 (PCB back, above PWR button) (fuel gauge)
  U9  xxpin  "2048, 33DH, X1MAQ" or so (small, below/right of PAIC) (accelero)
  U10  3pin  "HOX" or "XOH" (PCB back, below/right of YXAB) (magnet sensor?)
  U11 xxpin  unsharp... INVENSENSE or so (right of main cpu)  (gyroscope?)
  U12  6pin  "EPB" (between main cpu and wifi socket)
  U13  6pin  "?" (right of PAIC)
  Q1   6pin  "JG" (PCB back, right of POWER button)
  Q4   6pin  "?" (between PAIC and UC CTR)
  X1   4pin  "16.756" osc (near main CPU)
  X2   4pin  "CA038" osc (near UC CTR)
 Wifi board "DWM-W028"
  Un  80pin Wifi "Atheros, AR6014G.AL1C, N2T689,00B, 1036, TAIWAN" ;76+4pin
  Un   8pin Wifi SPI FLASH "32A, 0VX, 46"
  Un   8pin Wifi I2C EEPROM "08B, H1, 0DQ"
  Un   4pin voltage regulator? "M12, KA"
  Qn   4pin Crystal "40.000"
  Pn  50pin Connector to mainboard
  Pn   2pin Antenna connector
 IrDA board "CTR-IR-01 "
  U1   xpin IR photo led/transistor unit
  U2  24pin IrDA chip "NXP, S750, 0803, TSD031C"
 SD/MMC slot board
  Pn   Xpin Connector to mainboard
  Pn   Xpin Connector to full-size SD/MMC card
 Daughterboards:
  wifi-board, IrDA-board, sd slot, battery, etc - unknown
 XXXxxx...
  and probably much more unknown stuff...
```

3DS XL

2DS

New3DS

New2DS XL

Unknown. These consoles seem to be quite rare, and no PCB photos exist...?

New3DS LL Component List (aka japanese version of New3DS XL)

Mainboard (PCB "RED-CPU-01 SIDE B 11-1, [05: 14 071], RED-CPU-01 SIDE A")

```
Chips on Side B (top side, access requires removing mainboard):
Un  bga   CPU "1485 16, CPU LGR A, (C) 2014, Nintendo, JAPAN ARM"  ;\
Un  bga   RAM "F JAPAN, 82MK9A9A, 7L FCRAM, 1429969, E1"           ;
Un  6pin  MAX8570EUT+T for upper backlight "ABTJ"                  ; Side B
Un  sqr   Wifi "ATHEROS, AR6014G-AL1C, NKY197.00B, 1422, TAIWAN"   ;
Un  bga   eMMC "dotcode, SAMSUNG 410, KLM4G1YE0C-B301, CDA519GLN"  ;
Un  8pin  Wifi-EEPROM I2C "408F, B347" (HN58X2408F; 1Kx8)          ;
Un  bga   MCU "UC KTR, 423KM01, 'TK14"                             ;
Un  bga   Powerman? "TexasInstr, 93045A4, 3CAK08W L, GI"           ;/
Chips on Side A (bottom side, easily accessible):
Un  16pin BQ24072 battery charger "CKP, TI 3A1, CBYS" (near ext.supply input)
Un  6pin  power/charge? "9D" (or "06"?) (near ext.supply input)
Un  6pin  power "635Q" (under powerman) (...maybe lower backlight?)
Un  24pin IrDA "U8997, 3522" (nocash), or "NXP, S750, 1603, TSD438C" (ifixit)
Un  16pin Gyroscope? less-near infra red "IT3B,315BA1,LT425A" (like ITG-1010?)
Un  32pin NFC Broadcom BCM20791 "20791UZ, KMLG 25, TD1430, 3976901W"
Un  8pin  Wifi-FLASH SPI(?) "32B, 3XH, .01" (4Kx8) (near AIC)
Un  bga   AIC "TexasInstr, AIC3010D, 3BC473W, GI" (mic/sound/touchscr)
Un  16pin Accelerometer? near AIC and headphone "KXTKK, 40860, .3413" Kionix?
Un  bga   TCA6416A IO Expander "PH416A, TI 46T" ;near top-screen connectors
Other stuff:
Xn  4pin "CA405" or so (near UC KTR, probably for RTC)
Xn  4pin "40000, K42BY" (under atheros, probably for Wifi)
Xn  4pin "13000, K413Y" (under cpu, probably whatever) (to near-field chip?)
Xn  4pin "D164G" or so  (under cpu, probably whatever) (for CPU?)
Ix  Infrared receiver/transmitter
Fn  Fuse (for external charger + pin, near charger connector)
Pn  2pin External Power supply
Pn  3pin External Headphone
Pn  2pin Connector to antenna
Pn ??pin Connector to Button board (many pins)
Pn ??pin Connector to MicroSD slot (medium pins)
Pn  4pin Connector to Front panel Reset button? and MIC
Pn  4pin Connector to Front panel Home button? (4pins, only 2 wires attached)
Pn  Connector to bottom-shell (left shoulder (8pin+shield?, only 4 wires used)
Pn  Connector to bottom-shell (right shoulder (8pin+shld?, only 4 wires used)
Pn  4pin Connector to NFC (rectangle under bottom screen) (only 2 wires used)
Pn  4pin Connector to analog stick (above dpad)
Pn  4pin Connector to bottom screen, touchscreen
Pn  4pin Connector to bottom screen, backlight (4pin, only 2 wires used)
Pn ??pin Connector to bottom screen, video (many pins)
Pn ??pin Connector to top screen? ;\presumably video,backlight,cameras,sound
Pn ??pin Connector to top screen? ; (many pins on each connector)
Pn ??pin Connector to top screen? ;/
Pn  1pin Goldplate as additonal connection to cartridge board GND
Transistors/resistors/capacitors and the like
Barcode/sticker "22946012 Q01011K N"
```

Button board (PCB "F-7, PWB, /ABXY, /AU-C, /RED-01, CS, 01, 3814")

```
Un  44pin ?? Chip "428A2, HF374, 7NU9"  ;for C-stick and ZL/ZR-buttons?
Un  3pin "(M)UN" (large thing, right of start button) (magnet sensor?)
Un  8pin "7048, xxxxxx?" (tiny bga chip, right of select button) (fuel gauge)
Pn  Connector to mainboard
Pn  Connector to battery (3pin)
Pn  Connector to rubber nibble (C-Stick) (4pin)
Pn  Connector to shoulder (8pin)
Sw  Six buttons (A,B,X,Y,Sel,Start)
Sw  Front button (Power)
Fn  Fuse (for battery + pin, above wifi led)
Led Four Front LEDs (two single ones, and one LED-pair)
Led Notification LED (small 4pin RGB LED? under shoulder-connector)
Transistors/resistors and the like
Barcode/sticker "30946012 Z0101E9 X"
```

DPAD board (PCB "F-KEY 01, R, ZD, HF, 4354.")
```
Pn  Connector to cartridge slot board (5pin)
Sw  Four switches (Up,Down,Left,Right)
```
Cartridge Slot board (PCB "(DC)17P-01, C-3, 1 '14 12")
```
Pn  Cartridge Slot ("4081911")
Pn  Connector to mainboard
Pn  Connector to DPAD board (5pin)
Pn  Goldplate as additonal connection to mainboard GND
```
Battery:
```
Name "Nintendo SPR-003, Rechargeable Battery, 3.7V 1750mAh 6.5Wh"
Dotcode with text "MKH905D10"
Back side "Li-ion 00, SPR-A-BPAA-C1"
```
Bottom Screen:
```
Name "1912TSS140725, 21P4808L, B0014747QTFD" (4.18")
```
Top-screen unit
```
black ribbon cable: to camera unit      ;outer/rear cable
orange ribbon cable: to slider boards   ;middle cable
orange ribbon cable: to lcd screen      ;inner/front cable
red cable: to wifi antenna
1x  LCD with Dotcode/sticker "LAM049M003A1, 0020S4911848"
2x  2pin speaker's "G4830V23A" or "G4B30V23A" or so
1x  2pin wifi-antenna (nameless pcb)
1x  camera unit (with two pcbs, 3 cameras, and shielding, see below)
2x  slider boards (see below)
```
Left slider/speaker board:
```
1x  volume slider (9.8Kohm)
1x  wires to left speaker
1x  13pin connector to lcd screen (maybe parallax or whatever)
1x  4pin connector to lcd screen (maybe backlight and/or whatever)
```
Right slider/speaker board:
```
1x  3d slider (8.4Kohm)
1x  wires to right speaker
```
Front facing camera board:
```
two cameras, capacitors (C1..C12), FB1, FB2
```
Self facing camera board:
```
one camera, capacitors (C1..C5), FB1, R1, Q1, D1 (ir-led?)
```
Case/bottom
```
"(new), Nintendo, o3DS LL, (C)2014 Nintendo Made in China RED-001"
"[MIC/KS], EC-14016, RED-RB-JPN-C0"
"[R]007-AC0104, [T]D14-0144001, V(Ci Li-ion, (Nintendo)"
Barcode/sticker "QJF10203741 6"
```

3DS Chipset Pinouts

New Gyroscope (ITG-1010)

1	VDDIO	Digital I/O supply voltage
2	SCL/SCLK	I2C serial clock (SCL); SPI serial clock (SCLK)
3	SDA/SDI	I2C serial data (SDA); SPI serial data input (SDI)
4	AD0/SDO	I2C Slave Address LSB (AD0); SPI serial data output (SDO)
5	/CS	SPI chip select (Low=SPI mode, High=I2C mode)
6	RESV	Reserved. Connect to Ground
7	INT	Interrupt digital output (totem pole or open-drain)
8	FSYNC	Frame synchronization digital input. Connect to GND if not used
9-12	NC	Not internally connected, may be used for PCB trace routing
13	GND	Power supply ground
14	REGOUT	Regulator filter capacitor connection
15	RESV-G	Reserved. Connect to Ground
16	VDD	Power supply voltage

Old Gyroscope (ITG-3200)

```
1     CLKIN   Optional external ref clock input, connect to GND if unused
2-5   NC      Not internally connected, may be used for PCB trace routing
6-7   RESV    Reserved. Do not connect
8     VLOGIC  Digital IO supply voltage. VLOGIC must be = VDD at all times
9     AD0     I2C Slave Address LSB
10    REGOUT  Regulator filter capacitor connection
11    RESV-G  Reserved - Connect to ground.
12    INT     Interrupt digital output (totem pole or open-drain)
13    VDD     Power supply voltage
14-17 NC      Not internally connected. May be used for PCB trace routing
18    GND     Power supply ground
19    RESV    Reserved. Do not connect
20    CPOUT   Charge pump capacitor connection
21-22 RESV    Reserved. Do not connect
23    SCL     I2C serial clock
24    SDA     I2C serial data
```

Accelerometer (LIS331DLH)

```
1  Vdd_IO      Power supply for I/O pins
2  NC          Not connected
3  NC          Not connected
4  SCL,SPC     I2C.clock (or SPI.clock)
5  GND         0V supply
6  SDA,SDI,SDO I2C.data (or SPI.data.in, or 3-wire.data.out)
7  SA0,SDO     I2C.device.id (or SPI.data.out)
8  CS          Mode (high=I2C, low=SPI)
9  INT 2       Inertial interrupt 2
10 Reserved    Connect to GND
11 INT 1       Inertial interrupt 1
12 GND         0V supply
13 GND         0V supply
14 Vdd         Power supply
15 Reserved    Connect to Vdd
16 GND         0V supply
```

IrDA (NXP SC16IS750, 24pin HVQFN24 package)

```
1  I  /RESET     Reset
2  I  XTAL1      Crystal input, or external clock input
3  O  XTAL2      Crystal output, or clock output
4  -  Vdd        Supply
5  I  I2C,/SPI   Interface mode select
6  I  /CS,A0     SPI chip select, or I2C device id
7  I  SI,A1      SPI data in, or I2C device id
8  O  SO         SPI data out
9  I  SCL,SCLK   SPI clk, or I2C clk
10 IO SDA        I2C data
11 O  /IRQ       Interrupt
12 IO GPIO0      I/O
13 IO GPIO1      I/O
14 IO GPIO2      I/O
15 IO GPIO3      I/O
16 -  Vss        GND
17 IO GPIO4,/DSR I/O, or modem data set ready
18 IO GPIO5,/DTR I/O, or modem data terminal ready
19 IO GPIO6,/CD  I/O, or modem carrier detect
20 IO GPIO7,/RI  I/O, or modem ring indicator
21 O  /RTS       UART request to send
22 I  /CTS       UART clear to send
23 O  TX         UART data out
24 I  RX         UART data in
```

NFC (Broadcom BCM2079x)

Caution: below is for 34pin BCM20793S (which is NOT used in New3DS)

Caution: actual New3DS uses 32pin BCM20791 (maybe similar, but has only 32pin)
The pinout difference might be as simple as omitting the two N/C pins, or some of the four EEPROM/VDD_EE pins...?

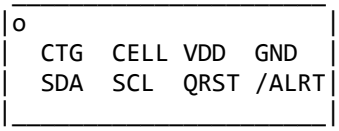
```
 1  VDDSWP_IN     Platform UICC supply in
 2  VDDSWP_OUT0   UICC supply out
 3  SWPIO_0       SWP I/O 0
 4  SWPIO_1       SWP I/O 1
 5  VDDSWP_OUT1   Supply to embedded secure element
 6  VDD_ADC       Decoupling, need linking to VDDC_CAP via target PCB
 7  NFC_WAKE      Signal from host to the BCM20793S
 8  TM2           ATE test mode (grounded for normal operation)
 9  VDD_EEPROM_IN 1.8V power input to the co-packaged EEPROM (get from VDD_EE)
 10 HOST_WAKE     Interrupt signal from the BCM20793S to host
 11 CLK_REQ       Clock request
 12 UART_TXD      UART transmit       ;or BSC/I2C_SDA, or SPI_MOSI
 13 VDDIO         I/O supply; externally regulated
 14 UART_RTS_N    UART ready to send  ;or BSC/I2C_SCL, or SPI_MISO
 15 SPI_INT       Host interface select (Low=UART, High=BSC/I2C, NC=SPI+IRQ)
 16 UART_CTS_N    UART clear to send  ;or BSC/I2C_REQ(?), or SPI_CS
 17 UART_RXD      UART receive        ;or SPI_CLK
 18 N/C           No connect
 19 XTAL_XON      Crystal N
 20 XTAL_XOP      Crystal P/clock reference input
 21 VDD_XTAL      Decoupling, need linking to VDDC_CAP via target PCB
 22 LPO           Frequency selection, strap high or low (see page 45)
 23 VDD_VCO       Decoupling, need linking to VDDC_CAP via target PCB
 24 VDDC_CAP      Decoupling, links VDDADC, VDD_XTAL, VDD_VCO via target PCB
 25 REG_PU        Regulator power control from host
 26 VBAT          Battery supply
 27 VDDA_CAP      Analog LDO supply decoupling (1.88V/2.5V)
 28 TX2           Coil output 2
 29 TX1           Coil output 1
 30 VDD_ANT       Rectifier output, external cap
 31 N/C           Do not connect
 32 VDD_EE        Output 1.8V supply voltage (to be linked to VDD_SE_IN)
 33 SDA (eeprom)  BSC/I2C data internally connected to co-packaged EEPROM
 34 SCL (eeprom)  BSC/I2C clock internally connected to co-packaged EEPROM
```

EEPROM: CAT24C64 EEPROM Die
Unknown if the EEPROM exists in New3DS, it might require REG_PU wake or so? Or the EEPROM is accessed INTERNALLY by the NFC chip (and the two EXTERNAL I2C/BSC EEPROM pins or meant to be used only optionally, for factory/init/test or so?) (that way the NFC could be also accessed via SPI/UART whilst internally talking to the I2C/BSC EEPROM).

MAX17040 Fuel Gauge (used in Old3DS)

```
 1 CTG   Connect to GND
 2 CELL  Battery Voltage Input
 3 VDD   Power Supply
 4 GND   Ground
 5 SEO   External 32kHz clock enable
 6 EO    External 32kHz clock
 7 SCL   I2C Clock
 8 SDA   I2C Data
```

MAX17048 Fuel Gauge (this, or something similar(?) is used in New3DS)



ARM CPU Reference

General ARM7TDMI Information
ARM CPU Overview
ARM CPU Register Set
ARM CPU Flags & Condition Field (cond)
ARM CPU 26bit Memory Interface
ARM CPU Exceptions
ARM CPU Memory Alignments

ARM 32bit Instruction Set (ARM Code)
ARM Instruction Summary
ARM Opcodes: Branch and Branch with Link (B, BL, BX, BLX, SWI, BKPT)
ARM Opcodes: Data Processing (ALU)
ARM Opcodes: Multiply and Multiply-Accumulate (MUL, MLA)
ARM Opcodes: Special ARM9 Instructions (CLZ, QADD/QSUB)
ARM Opcodes: Special ARM11 Instructions (Misc)
ARM Opcodes: Special ARM11 Instructions (SIMD)
ARM Opcodes: PSR Transfer (MRS, MSR)
ARM Opcodes: Memory: Single Data Transfer (LDR, STR, PLD)
ARM Opcodes: Memory: Halfword, Doubleword, and Signed Data Transfer
ARM Opcodes: Memory: Block Data Transfer (LDM, STM)
ARM Opcodes: Memory: Single Data Swap (SWP)
ARM Opcodes: Coprocessor Instructions (MRC/MCR, LDC/STC, CDP, MCRR/MRRC)

ARM 16bit Instruction Set (THUMB Code)
When operating in THUMB state, cut-down 16bit opcodes are used.
THUMB is supported on T-variants of ARMv4 and up, ie. ARMv4T, ARMv5T, etc.
THUMB Instruction Summary
THUMB Opcodes: Register Operations (ALU, BX)
THUMB Opcodes: Memory Load/Store (LDR/STR)
THUMB Opcodes: Memory Addressing (ADD PC/SP)
THUMB Opcodes: Memory Multiple Load/Store (PUSH/POP and LDM/STM)
THUMB Opcodes: Jumps and Calls
THUMB Opcodes: New THUMB Opcodes in ARM11

Note
Switching between ARM and THUMB state can be done by using the Branch and Exchange (BX) instruction.

Further Information
ARM Pseudo Instructions and Directives
ARM CP15 System Control Coprocessor
ARM Vector Floating-point Unit (VFP)
ARM CPU Instruction Cycle Times
ARM CPU Versions
ARM CPU Data Sheet

ARM CPU Overview

The ARM7TDMI is a 32bit RISC (Reduced Instruction Set Computer) CPU, designed by ARM (Advanced RISC Machines), and designed for both high performance and low power consumption.

Fast Execution
Depending on the CPU state, all opcodes are sized 32bit or 16bit (that's counting both the opcode bits and its parameters bits) providing fast decoding and execution. Additionally, pipelining allows - (a) one instruction to be executed while (b) the next instruction is decoded and (c) the next instruction is fetched from memory - all at the same time.

Data Formats
The CPU manages to deal with 8bit, 16bit, and 32bit data, that are called:

```
  8bit - Byte
 16bit - Halfword
 32bit - Word
```

The two CPU states
As mentioned above, two CPU states exist:
- ARM state: Uses the full 32bit instruction set (32bit opcodes)
- THUMB state: Uses a cutdown 16bit instruction set (16bit opcodes)
Regardless of the opcode-width, both states are using 32bit registers, allowing 32bit memory addressing as well as 32bit arithmetic/logical operations.

When to use ARM state
Basically, there are two advantages in ARM state:

```
- Each single opcode provides more functionality, resulting
  in faster execution when using a 32bit bus memory system
  (such like opcodes stored in GBA Work RAM).
- All registers R0-R15 can be accessed directly.
```

The downsides are:

```
- Not so fast when using 16bit memory system
  (but it still works though).
- Program code occupies more memory space.
```

When to use THUMB state
There are two major advantages in THUMB state:

```
- Faster execution up to approx 160% when using a 16bit bus
  memory system (such like opcodes stored in GBA GamePak ROM).
- Reduces code size, decreases memory overload down to approx 65%.
```

The disadvantages are:

```
- Not as multi-functional opcodes as in ARM state, so it will
  be sometimes required use more than one opcode to gain a
  similar result as for a single opcode in ARM state.
- Most opcodes allow only registers R0-R7 to be used directly.
```

Combining ARM and THUMB state
Switching between ARM and THUMB state is done by a normal branch (BX) instruction which takes only a handful of cycles to execute (allowing to change states as often as desired - with almost no overload).

Also, as both ARM and THUMB are using the same register set, it is possible to pass data between ARM and THUMB mode very easily.

The best memory & execution performance can be gained by combining both states: THUMB for normal program code, and ARM code for timing critical subroutines (such like interrupt handlers, or complicated algorithms).

Note: ARM and THUMB code cannot be executed simultaneously.

Automatic state changes
Beside for the above manual state switching by using BX instructions, the following situations involve automatic state changes:

- CPU switches to ARM state when executing an exception
- User switches back to old state when leaving an exception

ARM CPU Register Set

Overview
The following table shows the ARM7TDMI register set which is available in each mode. There's a total of 37 registers (32bit each), 31 general registers (Rxx) and 6 status registers (xPSR).
Note that only some registers are 'banked', for example, each mode has it's own R14 register: called R14, R14_fiq, R14_svc, etc. for each mode respectively.
However, other registers are not banked, for example, each mode is using the same R0 register, so writing to R0 will always affect the content of R0 in other modes also.

System/User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13 (SP)	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14 (LR)	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15	R15	R15	R15	R15
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
--	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

R0-R12 Registers (General Purpose Registers)
These thirteen registers may be used for whatever general purposes. Basically, each is having same functionality and performance, ie. there is no 'fast accumulator' for arithmetic operations, and no 'special pointer register' for memory addressing.
However, in THUMB mode only R0-R7 (Lo registers) may be accessed freely, while R8-R12 and up (Hi registers) can be accessed only by some instructions.

R13 Register (SP)
This register is used as Stack Pointer (SP) in THUMB state. While in ARM state the user may decided to use R13 and/or other register(s) as stack pointer(s), or as general purpose register.
As shown in the table above, there's a separate R13 register in each mode, and (when used as SP) each exception handler may (and MUST!) use its own stack.

R14 Register (LR)
This register is used as Link Register (LR). That is, when calling to a sub-routine by a Branch with Link (BL) instruction, then the return address (ie. old value of PC) is saved in this register.
Storing the return address in the LR register is obviously faster than pushing it into memory, however, as there's only one LR register for each mode, the user must manually push its content before issuing 'nested' subroutines.
Same happens when an exception is called, PC is saved in LR of new mode.

Note: In ARM mode, R14 may be used as general purpose register also, provided that above usage as LR register isn't required.

R15 Register (PC)
R15 is always used as program counter (PC). Note that when reading R15, this will usually return a value of PC+nn because of read-ahead (pipelining), whereas 'nn' depends on the instruction and on the CPU state (ARM or THUMB).

CPSR and SPSR (Program Status Registers) (ARMv3 and up)
The current condition codes (flags) and CPU control bits are stored in the CPSR register. When an exception arises, the old CPSR is saved in the SPSR of the respective exception-mode (much like PC is saved in LR).
For details refer to chapter about CPU Flags.

ARM CPU Flags & Condition Field (cond)

ARM Condition Field {cond}
The opcode {cond} suffixes can be used for conditionally executed code based on the C,N,Z,V flags in CPSR register. For example: BEQ = Branch if Equal, MOVMI = Move if Signed.
In ARM mode, {cond} can be used with all opcodes (except for a few newer ARMv5 instructions: BKPT, PLD, CDP2, LDC2, MCR2, MRC2, STC2, and BLX_imm are nonconditional; however BLX_reg can be conditional).
In THUMB mode, {cond} can be used only for branch opcodes.

Code	Suffix	Flags	Meaning
0:	EQ	Z=1	equal (zero) (same)
1:	NE	Z=0	not equal (nonzero) (not same)
2:	CS/HS	C=1	unsigned higher or same (carry set)
3:	CC/LO	C=0	unsigned lower (carry cleared)
4:	MI	N=1	signed negative (minus)
5:	PL	N=0	signed positive or zero (plus)
6:	VS	V=1	signed overflow (V set)
7:	VC	V=0	signed no overflow (V cleared)
8:	HI	C=1 and Z=0	unsigned higher
9:	LS	C=0 or Z=1	unsigned lower or same
A:	GE	N=V	signed greater or equal
B:	LT	N<>V	signed less than
C:	GT	Z=0 and N=V	signed greater than
D:	LE	Z=1 or N<>V	signed less or equal
E:	AL	-	always (the "AL" suffix can be omitted)
F:	NV	-	never (ARMv1,v2 only) (Reserved ARMv3 and up)

Execution Time: If condition=false: 1S cycle. Otherwise: as specified for the respective opcode.

Current Program Status Register (CPSR)

```
Bit   Expl.
31    N - Sign Flag       (0=Not Signed, 1=Signed)              ;\
30    Z - Zero Flag       (0=Not Zero, 1=Zero)                  ; Condition
29    C - Carry Flag      (0=Borrow/No Carry, 1=Carry/No Borrow) ; Code Flags
28    V - Overflow Flag   (0=No Overflow, 1=Overflow)           ;/
27    Q - Sticky Overflow (1=Sticky Overflow, ARMv5TE and up only)
26-8  Reserved            (For future use) - Do not change manually!
7     I - IRQ disable     (0=Enable, 1=Disable)                    ;\
6     F - FIQ disable     (0=Enable, 1=Disable)                    ; Control
5     T - State Bit       (0=ARM, 1=THUMB) - Do not change manually!; Bits
4-0   M4-M0 - Mode Bits   (See below)                              ;/
```

Bit 31-28: Condition Code Flags (N,Z,C,V)
These bits reflect results of logical or arithmetic instructions. In ARM mode, it is often optionally whether an instruction should modify flags or not, for example, it is possible to execute a SUB instruction that does NOT modify the condition flags.

In ARM state, all instructions can be executed conditionally depending on the settings of the flags, such like MOVEQ (Move if Z=1). While In THUMB state, only Branch instructions (jumps) can be made conditionally.

Bit 27: Sticky Overflow Flag (Q) - ARMv5TE and ARMv5TExP and up only
Used by QADD, QSUB, QDADD, QDSUB, SMLAxy, and SMLAWy only. These opcodes set the Q-flag in case of overflows, but leave it unchanged otherwise. The Q-flag can be tested/reset by MSR/MRS opcodes only.

Bit 27-8: Reserved Bits (except Bit 27 on ARMv5TE and up, see above)
These bits are reserved for possible future implementations. For best forwards compatibility, the user should never change the state of these bits, and should not expect these bits to be set to a specific value.

Bit 7-0: Control Bits (I,F,T,M4-M0)
These bits may change when an exception occurs. In privileged modes (non-user modes) they may be also changed manually.
The interrupt bits I and F are used to disable IRQ and FIQ interrupts respectively (a setting of "1" means disabled).
The T Bit signalizes the current state of the CPU (0=ARM, 1=THUMB), this bit should never be changed manually - instead, changing between ARM and THUMB state must be done by BX instructions.
The Mode Bits M4-M0 contain the current operating mode.

```
Binary Hex Dec  Expl.
0xx00b 00h 0  - Old User       ;\26bit Backward Compatibility modes
0xx01b 01h 1  - Old FIQ        ; (supported only on ARMv3, except ARMv3G,
0xx10b 02h 2  - Old IRQ        ; and on some non-T variants of ARMv4)
0xx11b 03h 3  - Old Supervisor ;/
10000b 10h 16 - User (non-privileged)
10001b 11h 17 - FIQ
10010b 12h 18 - IRQ
10011b 13h 19 - Supervisor (SWI)
10111b 17h 23 - Abort
11011b 1Bh 27 - Undefined
11111b 1Fh 31 - System (privileged 'User' mode) (ARMv4 and up)
```

Writing any other values into the Mode bits is not allowed.

Saved Program Status Registers (SPSR_<mode>)
Additionally to above CPSR, five Saved Program Status Registers exist:
SPSR_fiq, SPSR_svc, SPSR_abt, SPSR_irq, SPSR_und
Whenever the CPU enters an exception, the current status register (CPSR) is copied to the respective SPSR_<mode> register. Note that there is only one SPSR for each mode, so nested exceptions inside of the same mode are allowed only if the exception handler saves the content of SPSR in memory.
For example, for an IRQ exception: IRQ-mode is entered, and CPSR is copied to SPSR_irq. If the interrupt handler wants to enable nested IRQs, then it must first push SPSR_irq before doing so.

ARM CPU 26bit Memory Interface

The 26bit Memory Interface was used by ARMv1 and ARMv2. The 32bit interface is used by ARMv3 and newer, however, 26bit backward compatibility was included in all ARMv3 (except ARMv3G), and optionally in some non-T variants of ARMv4.

Format of R15 in 26bit Mode (Program Counter Register)

```
Bit   Name     Expl.
31-28 N,Z,C,V  Flags (Sign, Zero, Carry, Overflow)
27-26 I,F      Interrupt Disable bits (IRQ, FIQ) (1=Disable)
25-2  PC       Program Counter, 24bit, Step 4 (64M range)
1-0   M1,M0    Mode (0=User, 1=FIQ, 2=IRQ, 3=Supervisor)
```

Branches with +/-32M range wrap the PC register, and can reach all 64M memory.

Reading from R15
If R15 is specified in bit16-19 of an opcode, then NZCVIF and M0,1 are masked (zero), otherwise the full 32bits are used.

Writing to R15
ALU opcodes with S=1, and LDM opcodes with PSR=1 can write to all 32bits in R15 (in 26bit mode, that is allowed even in user mode, though it does then affect only NZCF, not the write protected IFMM bits ???), other opcodes which write to R15 will modify only the program counter bits. Also, special CMP/CMN/TST/TEQ{P} opcodes can be used to write to the PSR bits in R15 without modifying the PC bits.

Exceptions
SWIs, Reset, Data/Prefetch Aborts and Undefined instructions enter Supervisor mode. Interrupts enter IRQ and FIQ mode. Additionally, a special 26bit Address Exception exists, which enters Supervisor mode on accesses to memory addresses>=64M as follows:

```
R14_svc = PC ($+8, including old PSR bits)
M1,M0 = 11b = supervisor mode, F=same, I=1, PC=14h,
to continue at the fault location, return by SUBS PC,LR,8.
```

32bit CPUs with 26bit compatibility mode can be configured to switch into 32bit mode when encountering exceptions.

ARM CPU Exceptions

Exception Vectors
The following are the exception vectors in memory. That is, when an exception arises, CPU is switched into ARM state, and the program counter (PC) is loaded by the respective address.

Address	Prio	Exception	Mode on Entry	Interrupt Flags
BASE+00h	1	Reset	Supervisor (_svc)	I=1, F=1
BASE+04h	7	Undefined Instruction	Undefined (_und)	I=1, F=unchanged
BASE+08h	6	Software Interrupt (SWI)	Supervisor (_svc)	I=1, F=unchanged
BASE+0Ch	5	Prefetch Abort	Abort (_abt)	I=1, F=unchanged
BASE+10h	2	Data Abort	Abort (_abt)	I=1, F=unchanged
BASE+14h	??	Address Exceeds 26bit	Supervisor (_svc)	I=1, F=unchanged
BASE+18h	4	Normal Interrupt (IRQ)	IRQ (_irq)	I=1, F=unchanged
BASE+1Ch	3	Fast Interrupt (FIQ)	FIQ (_fiq)	I=1, F=1

BASE is normally 00000000h, but may be optionally FFFF0000h in some ARM CPUs. Priority for simultaneously occuring exceptions ranges from Prio=1=Highest to Prio=7=Lowest.
As there's only space for one ARM opcode at each of the above addresses, it'd be usually recommended to deposit a Branch opcode into each vector, which'd then redirect to the actual exception handlers address.

Actions performed by CPU when entering an exception

```
- R14_<new mode>=PC+nn   ;save old PC, ie. return address
- SPSR_<new mode>=CPSR   ;save old flags
- CPSR new T,M bits      ;set to T=0 (ARM state), and M4-0=new mode
- CPSR new I bit         ;IRQs disabled (I=1), done by ALL exceptions
- CPSR new F bit         ;FIQs disabled (F=1), done by Reset and FIQ only
- PC=exception_vector    ;see table above
```

Above "PC+nn" depends on the type of exception. Basically, in ARM state that nn-offset is caused by pipelining, and in THUMB state an identical ARM-style 'offset' is generated (even though the 'base address' may be only halfword-aligned).

Required user-handler actions when returning from an exception
Restore any general registers (R0-R14) which might have been modified by the exception handler. Use return-instruction as listed in the respective descriptions below, this will both restore PC and CPSR - that automatically

involves that the old CPU state (THUMB or ARM) as well as old state of FIQ and IRQ disable flags are restored.
As mentioned above (see action on entering...), the return address is always saved in ARM-style format, so that exception handler may use the same return-instruction, regardless of whether the exception has been generated from inside of ARM or THUMB state.

FIQ (Fast Interrupt Request)
This interrupt is generated by a LOW level on the nFIQ input. It is supposed to process timing critical interrupts at a high priority, as fast as possible.
Additionally to the common banked registers (R13_fiq,R14_fiq), five extra banked registers (R8_fiq-R12_fiq) are available in FIQ mode. The exception handler may freely access these registers without modifying the main programs R8-R12 registers (and without having to save that registers on stack).
In privileged (non-user) modes, FIQs may be also manually disabled by setting the F Bit in CPSR.

IRQ (Normal Interrupt Request)
This interrupt is generated by a LOW level on the nIRQ input. Unlike FIQ, the IRQ mode is not having its own banked R8-R12 registers.
IRQ is having lower priority than FIQ, and IRQs are automatically disabled when a FIQ exception becomes executed. In privileged (non-user) modes, IRQs may be also manually disabled by setting the I Bit in CPSR.
To return from IRQ Mode (continuing at following opcode):
```
  SUBS PC,R14,4   ;both PC=R14_irq-4, and CPSR=SPSR_irq
```

Software Interrupt
Generated by a software interrupt instruction (SWI). Recommended to request a supervisor (operating system) function. The SWI instruction may also contain a parameter in the 'comment field' of the opcode:
In case that your main program issues SWIs from both inside of THUMB and ARM states, then your exception handler must separate between 24bit comment fields in ARM opcodes, and 8bit comment fields in THUMB opcodes (if necessary determine old state by examining T Bit in SPSR_svc); However, in Little Endian mode, you could use only the most significant 8bits of the 24bit ARM comment field (as done in the GBA, for example) - the exception handler could then process the BYTE at [R14-2], regardless of whether it's been called from ARM or THUMB state.
To return from Supervisor Mode (continuing at following opcode):
```
  MOVS PC,R14   ;both PC=R14_svc, and CPSR=SPSR_svc
```
Note: Like all other exceptions, SWIs are always executed in ARM state, no matter whether it's been caused by an ARM or THUMB state SWI instruction.

Undefined Instruction Exception (supported by ARMv3 and up)
This exception is generated when the CPU comes across an instruction which it cannot handle. Most likely signalizing that the program has locked up, and that an errormessage should be displayed.
However, it might be also used to emulate custom functions, ie. as an additional 'SWI' instruction (which'd use R14_und and SPSR_und though, and it'd thus allow to execute the Undefined Instruction handler from inside of Supervisor mode without having to save R14_svc and SPSR_svc).
To return from Undefined Mode (continuing at following opcode):
```
  MOVS PC,R14   ;both PC=R14_und, and CPSR=SPSR_und
```
Note that not all unused opcodes are necessarily producing an exception, for example, an ARM state Multiply instruction with Bit6=1 would be blindly accepted as 'legal' opcode.

Abort (supported by ARMv3 and up)
Aborts (page faults) are mostly supposed for virtual memory systems (ie. not used in GBA, as far as I know), otherwise they might be used just to display an error message. Two types of aborts exists:
- Prefetch Abort (occurs during an instruction prefetch)
- Prefetch Abort (also occurs on BKPT opcodes, ARMv5 and up)
- Data Abort (occurs during a data access)

A virtual memory systems abort handler would then most likely determine the fault address: For prefetch abort

that's just "R14_abt-4". For Data abort, the THUMB or ARM instruction at "R14_abt-8" needs to be 'disassembled' in order to determine the addressed data in memory.
The handler would then fix the error by loading the respective memory page into physical memory, and then retry to execute the SAME instruction again, by returning as follows:

```
prefetch abort: SUBS PC,R14,#4   ;PC=R14_abt-4, and CPSR=SPSR_abt
data abort:     SUBS PC,R14,#8   ;PC=R14_abt-8, and CPSR=SPSR_abt
```

Separate exception vectors for prefetch/data abort exists, each should use the respective return instruction as shown above.

Address Exceeds 26bit
This exception can occur only on old ARM CPUs with 26bit address scheme (or in 26bit backwards compatibility mode).

Reset
Forces PC=VVVV0000h, and forces control bits of CPSR to T=0 (ARM state), F=1 and I=1 (disable FIQ and IRQ), and M4-0=10011b (Supervisor mode).

ARM CPU Memory Alignments

The CPU does NOT support accessing mis-aligned addresses (which would be rather slow because it'd have to merge/split that data into two accesses).
When reading/writing code/data to/from memory, Words and Halfwords must be located at well-aligned memory address, ie. 32bit words aligned by 4, and 16bit halfwords aligned by 2.

Mis-aligned STR,STRH,STM,LDM,LDRD,STRD,PUSH,POP (forced align)
The mis-aligned low bit(s) are ignored, the memory access goes to a forcibly aligned (rounded-down) memory address.
For LDRD/STRD, it isn't clearly defined if the address must be aligned by 8 (on the NDS, align-4 seems to be okay) (align-8 may be required on other CPUs with 64bit databus).

Mis-aligned LDR,SWP (rotated read)
Reads from forcibly aligned address "addr AND (NOT 3)", and does then rotate the data as "ROR (addr AND 3)*8". That effect is internally used by LDRB and LDRH opcodes (which do then mask-out the unused bits).
The SWP opcode works like a combination of LDR and STR, that means, it does read-rotated, but does write-unrotated.

Mis-aligned LDRH,LDRSH (does or does not do strange things)
On ARM9 aka ARMv5 aka NDS9:

```
LDRH Rd,[odd]   -->  LDRH Rd,[odd-1]        ;forced align
LDRSH Rd,[odd]  -->  LDRSH Rd,[odd-1]       ;forced align
```

On ARM7 aka ARMv4 aka NDS7/GBA:

```
LDRH Rd,[odd]   -->  LDRH Rd,[odd-1] ROR 8  ;read to bit0-7 and bit24-31
LDRSH Rd,[odd]  -->  LDRSB Rd,[odd]         ;sign-expand BYTE value
```

Mis-aligned PC/R15 (branch opcodes, or MOV/ALU/LDR with Rd=R15)
For ARM code, the low bits of the target address should be usually zero, otherwise, R15 is forcibly aligned by clearing the lower two bits.
For THUMB code, the low bit of the target address may/should/must be set, the bit is (or is not) interpreted as thumb-bit (depending on the opcode), and R15 is then forcibly aligned by clearing the lower bit.
In short, R15 will be always forcibly aligned, so mis-aligned branches won't have effect on subsequent opcodes that use R15, or [R15+disp] as operand.

ARM Instruction Summary

Modification of CPSR flags is optional for all {S} instructions.

Logical ALU Operations

Instruction	Cycles	Flags	Expl.
MOV{cond}{S} Rd,Op2	1S+x+y	NZc-	Rd = Op2
MVN{cond}{S} Rd,Op2	1S+x+y	NZc-	Rd = NOT Op2
ORR{cond}{S} Rd,Rn,Op2	1S+x+y	NZc-	Rd = Rn OR Op2
EOR{cond}{S} Rd,Rn,Op2	1S+x+y	NZc-	Rd = Rn XOR Op2
AND{cond}{S} Rd,Rn,Op2	1S+x+y	NZc-	Rd = Rn AND Op2
BIC{cond}{S} Rd,Rn,Op2	1S+x+y	NZc-	Rd = Rn AND NOT Op2
TST{cond}{P} Rn,Op2	1S+x	NZc-	Void = Rn AND Op2
TEQ{cond}{P} Rn,Op2	1S+x	NZc-	Void = Rn XOR Op2

Add x=1I cycles if Op2 shifted-by-register. Add y=1S+1N cycles if Rd=R15.
Carry flag affected only if Op2 contains a non-zero shift amount.

Arithmetic ALU Operations

Instruction	Cycles	Flags	Expl.
ADD{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Rn+Op2
ADC{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Rn+Op2+Cy
SUB{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Rn-Op2
SBC{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Rn-Op2+Cy-1
RSB{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Op2-Rn
RSC{cond}{S} Rd,Rn,Op2	1S+x+y	NZCV	Rd = Op2-Rn+Cy-1
CMP{cond}{P} Rn,Op2	1S+x	NZCV	Void = Rn-Op2
CMN{cond}{P} Rn,Op2	1S+x	NZCV	Void = Rn+Op2

Add x=1I cycles if Op2 shifted-by-register. Add y=1S+1N cycles if Rd=R15.

Multiply

Instruction	Cycles	Flags	Expl.
MUL{cond}{S} Rd,Rm,Rs	1S+mI	NZx-	Rd = Rm*Rs
MLA{cond}{S} Rd,Rm,Rs,Rn	1S+mI+1I	NZx-	Rd = Rm*Rs+Rn
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	1S+mI+1I	NZx-	RdHiLo = Rm*Rs
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	1S+mI+2I	NZx-	RdHiLo = Rm*Rs+RdHiLo
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	1S+mI+1I	NZx-	RdHiLo = Rm*Rs
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	1S+mI+2I	NZx-	RdHiLo = Rm*Rs+RdHiLo
SMLAxy{cond} Rd,Rm,Rs,Rn	ARMv5TE(xP)	----q	Rd=HalfRm*HalfRs+Rn
SMLAWy{cond} Rd,Rm,Rs,Rn	ARMv5TE(xP)	----q	Rd=(Rm*HalfRs)/10000h+Rn
SMULWy{cond} Rd,Rm,Rs	ARMv5TE(xP)	----	Rd=(Rm*HalfRs)/10000h
SMLALxy{cond} RdLo,RdHi,Rm,Rs	ARMv5TE(xP)	----	RdHiLo=RdHiLo+HalfRm*HalfRs
SMULxy{cond} Rd,Rm,Rs	ARMv5TE(xP)	----	Rd=HalfRm*HalfRs

Memory Load/Store

Instruction	Cycles	Flags	Expl.
LDR{cond}{B}{T} Rd,<Address>	1S+1N+1I+y	----	Rd=[Rn+/-<offset>]
LDR{cond}H Rd,<Address>	1S+1N+1I+y	----	Load Unsigned halfword
LDR{cond}D Rd,<Address>		----	Load Dword ARMv5TE
LDR{cond}SB Rd,<Address>	1S+1N+1I+y	----	Load Signed byte
LDR{cond}SH Rd,<Address>	1S+1N+1I+y	----	Load Signed halfword
LDM{cond}{amod} Rn{!},<Rlist>{^}	nS+1N+1I+y	----	Load Multiple
STR{cond}{B}{T} Rd,<Address>	2N	----	[Rn+/-<offset>]=Rd
STR{cond}H Rd,<Address>	2N	----	Store halfword
STR{cond}D Rd,<Address>		----	Store Dword ARMv5TE
STM{cond}{amod} Rn{!},<Rlist>{^}	(n-1)S+2N	----	Store Multiple
SWP{cond}{B} Rd,Rm,[Rn]	1S+2N+1I	----	Rd=[Rn], [Rn]=Rm
PLD <Address>	1S	----	Prepare Cache ARMv5TE

For LDR/LDM, add y=1S+1N if Rd=R15, or if R15 in Rlist.

Jumps, Calls, CPSR Mode, and others

Instruction	Cycles	Flags	Expl.

```
B{cond}   label              2S+1N    ----  PC=$+8+/-32M
BL{cond}  label              2S+1N    ----  PC=$+8+/-32M, LR=$+4
BX{cond}  Rn                 2S+1N    ----  PC=Rn, T=Rn.0 (THUMB/ARM)
BLX{cond} Rn                 2S+1N    ----  PC=Rn, T=Rn.0, LR=PC+4, ARM9
BLX       label              2S+1N    ----  PC=PC+$+/-32M, LR=$+4, T=1, ARM9
MRS{cond} Rd,Psr             1S       ----  Rd=Psr
MSR{cond} Psr{_field},Op     1S     (psr)  Psr[field]=Op
SWI{cond} Imm24bit           2S+1N    ----  PC=8, ARM Svc mode, LR=$+4
BKPT      Imm16bit           ???      ----  PC=C, ARM Abt mode, LR=$+4 ARM9
The Undefined Instruction    2S+1I+1N ----  PC=4, ARM Und mode, LR=$+4
cond=false                   1S       ----  Any opcode with condition=false
NOP                          1S       ----  R0=R0

CLZ{cond} Rd,Rm              ???      ----    Count Leading Zeros ARMv5
QADD{cond} Rd,Rm,Rn                   ----q   Rd=Rm+Rn       ARMv5TE(xP)
QSUB{cond} Rd,Rm,Rn                   ----q   Rd=Rm-Rn       ARMv5TE(xP)
QDADD{cond} Rd,Rm,Rn                  ----q   Rd=Rm+Rn*2     ARMv5TE(xP)
QDSUB{cond} Rd,Rm,Rn                  ----q   Rd=Rm-Rn*2     ARMv5TE(xP)
```

Coprocessor Functions (if any)

```
Instruction                          Cycles  Flags  Expl.
CDP{cond} Pn,<cpopc>,Cd,Cn,Cm{,<cp>} 1S+bI   ----  Coprocessor specific
STC{cond}{L} Pn,Cd,<Address>         (n-1)S+2N+bI  [address] = CRd
LDC{cond}{L} Pn,Cd,<Address>         (n-1)S+2N+bI  CRd = [address]
MCR{cond} Pn,<cpopc>,Rd,Cn,Cm{,<cp>} 1S+bI+1C      CRn = Rn {<op> CRm}
MRC{cond} Pn,<cpopc>,Rd,Cn,Cm{,<cp>} 1S+(b+1)I+1C  Rn = CRn {<op> CRm}
CDP2,STC2,LDC2,MCR2,MRC2 - ARMv5 Extensions similar above, without {cond}
MCRR{cond} Pn,<cpopc>,Rd,Rn,Cm  ;write Rd,Rn to coproc ARMv5TE
MRRC{cond} Pn,<cpopc>,Rd,Rn,Cm  ;read Rd,Rn from coproc ARMv5TE
```

ARM Binary Opcode Format

```
|..3 .................2 .................1 .................0|
|1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0|
|_Cond__|0_0_0|___Op__|S|__Rn___|__Rd___|__Shift__|Typ|0|__Rm___| DataProc
|_Cond__|0_0_0|___Op__|S|__Rn___|__Rd___|__Rs___|0|Typ|1|__Rm___| DataProc
|_Cond__|0_0_1|___Op__|S|__Rn___|__Rd___|_Shift_|___Immediate___| DataProc
|_Cond__|0_0_1_1_0_0_1_0_0_0_0_0_1_1_1_1_0_0_0_0|_____Hint______| ARM11:Hint
|_Cond__|0_0_1_1_0|P|1|0|_Field_|__Rd___|_Shift_|___Immediate___| PSR Imm
|_Cond__|0_0_0_1_0|P|L|0|_Field_|__Rd___|0_0_0_0|0_0_0_0|__Rm___| PSR Reg
|_Cond__|0_0_0_1_0_0_1_0_1_1_1_1_1_1_1_1_1_1_1_1|0_0|L|1|__Rn___| BX,BLX
|1_1_1_0|0_0_0_1_0_0_1_0|_____immediate_________|0_1_1_1|_immed_| ARM9:BKPT
|_Cond__|0_0_0_1_0_1_1_0_1_1_1_1|__Rd___|1_1_1_1|0_0_0_1|__Rm___| ARM9:CLZ
|_Cond__|0_0_0_1_0|Op_|0|__Rn___|__Rd___|0_0_0_0|0_1_0_1|__Rm___| ARM9:QALU
|_Cond__|0_0_0_0_0_0|A|S|__Rd___|__Rn___|__Rs___|1_0_0_1|__Rm___| Multiply
|_Cond__|0_0_0_0_0_1_0_0|_RdHi__|_RdLo__|__Rs___|1_0_0_1|__Rm___| ARM11:UMAAL
|_Cond__|0_0_0_0_1|U|A|S|_RdHi__|_RdLo__|__Rs___|1_0_0_1|__Rm___| MulLong
|_Cond__|0_0_0_1_0|Op_|0|Rd/RdHi|Rn/RdLo|__Rs___|1|y|x|0|__Rm___| MulHalfARM9
|_Cond__|0_0_0_1_0|B|0_0|__Rn___|__Rd___|0_0_0_0|1_0_0_1|__Rm___| TransSwp12
|_Cond__|0_0_0_1_1|_Op__|__Rn___|__Rd___|1_1_1_1|1_0_0_1|__Rm___| ARM11:LDREX
|_Cond__|0_0_0|P|U|0|W|L|__Rn___|__Rd___|0_0_0_0|1|S|H|1|__Rm___| TransReg10
|_Cond__|0_0_0|P|U|1|W|L|__Rn___|__Rd___|OffsetH|1|S|H|1|OffsetL| TransImm10
|_Cond__|0_1_0|P|U|B|W|L|__Rn___|__Rd___|________Offset________| TransImm9
|_Cond__|0_1_1|P|U|B|W|L|__Rn___|__Rd___|__Shift__|Typ|0|__Rm___| TransReg9
|_Cond__|0_1_1|_____________xxx_____________________|1|__xxx__| Undefined
|_Cond__|0_1_1|Op_|x_x_x_x_x_x_x_x_x_x_x_x_x_x_x_x_x_x_x_x|1|x_x_x_x| ARM11:Media
|1_1_1_1_0_1_0_1_0_1_1_1_1_1_1_1_1_1_1_1_0_0_0_0_0_0_0_1_1_1_1_1| ARM11:CLREX
|_Cond__|1_0_0|P|U|S|W|L|__Rn___|_________Register_List________| BlockTrans
|_Cond__|1_0_1|L|___________________Offset____________________| B,BL,BLX
|_Cond__|1_1_0|P|U|N|W|L|__Rn___|__CRd__|__CP#__|____Offset_____| CoDataTrans
|_Cond__|1_1_0_0_0_1_0|L|__Rn___|__Rd___|__CP#__|_CPopc_|__CRm__| CoRR ARM9
|_Cond__|1_1_1_0|_CPopc_|__CRn__|__CRd__|__CP#__|_CP__|0|__CRm__| CoDataOp
|_Cond__|1_1_1_0|CPopc|L|__CRn__|__Rd___|__CP#__|_CP__|1|__CRm__| CoRegTrans
|_Cond__|1_1_1_1|_____________Ignored_by_Processor_____________| SWI
```

ARM11 SIMD Media instructions (ARM11/ARMv6) (formerly Undef opcode space)

```
  |..3 .................2 .................1 .................0|
  |1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0|
  |_Cond__|0_1_1|0_0|_Op__|__Rn___|__Rd___|1_1_1_1|_Op__|1|__Rm___| ParaAddSub
  |_Cond__|0_1_1|0_1|0_0_0|__Rn___|__Rd___|_shift___|x|0_1|__Rm___| HalfPack
  |_Cond__|0_1_1|0_1|U|1|____sat__|__Rd___|_shift___|x|0_1|__Rm___| WordSat
  |_Cond__|0_1_1|0_1|U|1_0|__sat__|__Rd___|1_1_1_1|0_0_1_1|__Rm___| ParaHalfSat
  |_Cond__|0_1_1|0_1|0_1_1|1_1_1_1|__Rd___|1_1_1_1|0_0_1_1|__Rm___| RevWord
  |_Cond__|0_1_1|0_1|0_1_1|1_1_1_1|__Rd___|1_1_1_1|1_0_1_1|__Rm___| RevPackHalf
  |_Cond__|0_1_1|0_1|1_1_1|1_1_1_1|__Rd___|1_1_1_1|1_0_1_1|__Rm___| RevSignHalf
  |_Cond__|0_1_1|0_1|0_0_0|__Rn___|__Rd___|1_1_1_1|1_0_1_1|__Rm___| SelectBytes
  |_Cond__|0_1_1|0_1|_Op__|__Rn___|__Rd___|rot|0_0|0_1_1_1|__Rm___| Extend U/S
  |_Cond__|0_1_1|1_0|_Op__|Rd/RdHi|Rn/RdLo|__Rs___|_Op__|1|__Rm___| Multiplies
  |_Cond__|0_1_1|1_1|0_0_0|__Rd___|__Rn___|__Rs___|0_0_0_1|__Rm___| DiffSum
  |_Cond__|0_1_1|1_1|0_0_0|__Rd___|1_1_1_1|__Rs___|0_0_0_1|__Rm___| DiffSumAcc
  |_Cond__|0_1_1_1_1_1_1_1|x_x_x_x_x_x_x_x_x_x_x_x|1_1_1_1|x_x_x_x| Undef
```

ARM Opcodes: Branch and Branch with Link (B, BL, BX, BLX, SWI, BKPT)

Branch and Branch with Link (B, BL, BLX_imm)

Branch (B) is supposed to jump to a subroutine. Branch with Link is meant to be used to call to a subroutine, return address is then saved in R14.

```
  Bit    Expl.
  31-28  Condition (must be 1111b for BLX)
  27-25  Must be "101" for this instruction
  24     Opcode (0-1) (or Halfword Offset for BLX)
          0: B{cond} label    ;branch            PC=PC+8+nn*4
          1: BL{cond} label   ;branch/link       PC=PC+8+nn*4, LR=PC+4
          H: BLX label ;ARM9  ;branch/link/thumb PC=PC+8+nn*4+H*2, LR=PC+4, T=1
  23-0   nn - Signed Offset, step 4      (-32M..+32M in steps of 4)
```

Branch with Link can be used to 'call' to a sub-routine, which may then 'return' by MOV PC,R14 for example.
Execution Time: 2S + 1N
Return: No flags affected.

Branch and Exchange (BX, BLX_reg)

```
  Bit    Expl.
  31-28  Condition
  27-8   Must be "0001.0010.1111.1111.1111" for this instruction
  7-4    Opcode
          0001b: BX{cond}  Rn    ;PC=Rn, T=Rn.0   (ARMv4T and ARMv5 and up)
          0010b: BXJ{cond} Rn    ;Change to Jazelle bytecode (ARMv5TEJ and up)
          0011b: BLX{cond} Rn    ;PC=Rn, T=Rn.0, LR=PC+4     (ARMv5 and up)
  3-0    Rn - Operand Register  (R0-R14)
```

Switching to THUMB Mode: Set Bit 0 of the value in Rn to 1, program continues then at Rn-1 in THUMB mode.
Using BLX R14 is possible (sets PC=Old_LR, and New_LR=retadr).
Using BX R15 acts as BX $+8 (tested and working on ARM7/ARM9, although it isn't officially defined as predictable behaviour).
Execution Time: 2S + 1N
Return: No flags affected.

Notes on BXJ (Branch or change to Jazelle bytecode state):

If Jazelle is suppored/enabled: Executes Java bytecode at address R14
If Jazelle is NOT suppored/enabled: Behaves exactly same as BX Rn
That is, R14 is the byte code address, and Rn is an ARM/THUMB error handler.

Supported on ARMv5TEJ and ARMv6 and up.

Branch via ALU, LDR, LDM
Most ALU, LDR, LDM opcodes can also change PC/R15.

Software Interrupt (SWI/BKPT) (svc/abt exceptions)
SWI supposed for calls to the operating system - Enter Supervisor mode (SVC) in ARM state. BKPT intended for debugging - enters Abort mode in ARM state via Prefetch Abort vector.

```
Bit    Expl.
31-28  Condition (must be 1110b for BKPT, ie. Condition=always)
27-24  Opcode
        1111b: SWI{cond} nn   ;software interrupt
        0001b: BKPT      nn   ;breakpoint (ARMv5 and up)
For SWI:
 23-0   nn - Comment Field, ignored by processor (24bit value)
For BKPT:
 23-20  Must be 0010b for BKPT
 19-8   nn - upper 12bits of comment field, ignored by processor
 7-4    Must be 0111b for BKPT
 3-0    nn - lower 4bits of comment field, ignored by processor
```

Execution Time: 2S+1N
The exception handler may interprete the SWI Comment Field by examining the lower 24bit of the 32bit opcode opcode at [R14_svc-4]. If your are also using SWI's from inside of THUMB, then the SWI handler must examine the T Bit SPSR_svc in order to determine whether it's been a THUMB SWI - and if so, examine the lower 8bit of the 16bit opcode opcode at [R14_svc-2].
For Returning from SWI use "MOVS PC,R14", that instruction does restore both PC and CPSR, ie. PC=R14_svc, and CPSR=SPSR_svc.
Nesting SWIs: SPSR_svc and R14_svc should be saved on stack before either invoking nested SWIs, or (if the IRQ handler uses SWIs) before enabling IRQs.
Execution SWI/BKPT:

```
R14_svc=PC+4     R14_abt=PC+4   ;save return address
SPSR_svc=CPSR    SPSR_abt=CPSR  ;save CPSR flags
CPSR=<changed>   CPSR=<changed> ;Enter svc/abt, ARM state, IRQs disabled
PC=VVVV0008h     PC=VVVV000Ch   ;jump to SWI/PrefetchAbort vector address
```

Undefined Instruction (und exception)

```
Bit    Expl.
31-28  Condition
27-25  Must be 011b for this instruction
24-5   Reserved for future use
4      Must be 1b for this instruction
3-0    Reserved for future use
```

No assembler mnemonic exists, following bitstreams are (not) reserved.

```
cond011xxxxxxxxxxxxxxxxxxxx1xxxx - reserved for future use (except below).
cond01111111xxxxxxxxxxxx1111xxxx - free for user.
```

Note: ARMv6 does use the "reserved for future" space for the new SIMD opcodes (but still has the "free for user" space available as "undefined").
Execution time: 2S+1I+1N.

ARM Opcodes: Data Processing (ALU)

Opcode Format

```
Bit    Expl.
31-28  Condition
27-26  Must be 00b for this instruction
25     I - Immediate 2nd Operand Flag (0=Register, 1=Immediate)
24-21  Opcode (0-Fh)              ;*=Arithmetic, otherwise Logical
```

```
          0: AND{cond}{S} Rd,Rn,Op2    ;AND logical       Rd = Rn AND Op2
          1: EOR{cond}{S} Rd,Rn,Op2    ;XOR logical       Rd = Rn XOR Op2
          2: SUB{cond}{S} Rd,Rn,Op2 ;* ;subtract          Rd = Rn-Op2
          3: RSB{cond}{S} Rd,Rn,Op2 ;* ;subtract reversed Rd = Op2-Rn
          4: ADD{cond}{S} Rd,Rn,Op2 ;* ;add               Rd = Rn+Op2
          5: ADC{cond}{S} Rd,Rn,Op2 ;* ;add with carry    Rd = Rn+Op2+Cy
          6: SBC{cond}{S} Rd,Rn,Op2 ;* ;sub with carry    Rd = Rn-Op2+Cy-1
          7: RSC{cond}{S} Rd,Rn,Op2 ;* ;sub cy. reversed  Rd = Op2-Rn+Cy-1
          8: TST{cond}{P}    Rn,Op2    ;test            Void = Rn AND Op2
          9: TEQ{cond}{P}    Rn,Op2    ;test exclusive  Void = Rn XOR Op2
          A: CMP{cond}{P}    Rn,Op2 ;* ;compare         Void = Rn-Op2
          B: CMN{cond}{P}    Rn,Op2 ;* ;compare neg.    Void = Rn+Op2
          C: ORR{cond}{S} Rd,Rn,Op2    ;OR logical        Rd = Rn OR Op2
          D: MOV{cond}{S} Rd,Op2       ;move              Rd = Op2
          E: BIC{cond}{S} Rd,Rn,Op2    ;bit clear         Rd = Rn AND NOT Op2
          F: MVN{cond}{S} Rd,Op2       ;not               Rd = NOT Op2
  20     S - Set Condition Codes (0=No, 1=Yes) (Must be 1 for opcode 8-B)
  19-16  Rn - 1st Operand Register (R0..R15) (including PC=R15)
             Must be 0000b for MOV/MVN.
  15-12  Rd - Destination Register (R0..R15) (including PC=R15)
             Must be 0000b (or 1111b) for CMP/CMN/TST/TEQ{P}.
  When above Bit 25 I=0 (Register as 2nd Operand)
    When below Bit 4 R=0 - Shift by Immediate
      11-7   Is - Shift amount   (1-31, 0=Special/See below)
    When below Bit 4 R=1 - Shift by Register
      11-8   Rs - Shift register (R0-R14) - only lower 8bit 0-255 used
      7      Reserved, must be zero  (otherwise multiply or LDREX or undefined)
    6-5    Shift Type (0=LSL, 1=LSR, 2=ASR, 3=ROR)
    4      R - Shift by Register Flag (0=Immediate, 1=Register)
    3-0    Rm - 2nd Operand Register (R0..R15) (including PC=R15)
  When above Bit 25 I=1 (Immediate as 2nd Operand)
    11-8   Is - ROR-Shift applied to nn (0-30, in steps of 2)
    7-0    nn - 2nd Operand Unsigned 8bit Immediate
```

Second Operand (Op2)
This may be a shifted register, or a shifted immediate. See Bit 25 and 11-0.
Unshifted Register: Specify Op2 as "Rm", assembler converts to "Rm,LSL#0".
Shifted Register: Specify as "Rm,SSS#Is" or "Rm,SSS Rs" (SSS=LSL/LSR/ASR/ROR).
Immediate: Specify as 32bit value, for example: "#000NN000h", assembler should automatically convert into "#0NNh,ROR#0ssh" as far as possible (ie. as far as a section of not more than 8bits of the immediate is non-zero).

Zero Shift Amount (Shift Register by Immediate, with Immediate=0)
```
  LSL#0: No shift performed, ie. directly Op2=Rm, the C flag is NOT affected.
  LSR#0: Interpreted as LSR#32, ie. Op2 becomes zero, C becomes Bit 31 of Rm.
  ASR#0: Interpreted as ASR#32, ie. Op2 and C are filled by Bit 31 of Rm.
  ROR#0: Interpreted as RRX#1 (RCR), like ROR#1, but Op2 Bit 31 set to old C.
```
In source code, LSR#32, ASR#32, and RRX#1 should be specified as such - attempts to specify LSR#0, ASR#0, or ROR#0 will be internally converted to LSL#0 by the assembler.

Using R15 (PC)
When using R15 as Destination (Rd), note below CPSR description and Execution time description.
When using R15 as operand (Rm or Rn), the returned value depends on the instruction: PC+12 if I=0,R=1 (shift by register), otherwise PC+8 (shift by immediate).

Returned CPSR Flags
If S=1, Rd<>R15, logical operations (AND,EOR,TST,TEQ,ORR,MOV,BIC,MVN):
```
  V=not affected
  C=carryflag of shift operation (not affected if LSL#0 or Rs=00h)
  Z=zeroflag of result
  N=signflag of result (result bit 31)
```

If S=1, Rd<>R15, arithmetic operations (SUB,RSB,ADD,ADC,SBC,RSC,CMP,CMN):
```
  V=overflowflag of result
  C=carryflag of result
  Z=zeroflag of result
  N=signflag of result (result bit 31)
```
IF S=1, with unused Rd bits=1111b, {P} opcodes (CMPP/CMNP/TSTP/TEQP):
```
  R15=result  ;modify PSR bits in R15, ARMv2 and below only.
  In user mode only N,Z,C,V bits of R15 can be changed.
  In other modes additionally I,F,M1,M0 can be changed.
  The PC bits in R15 are left unchanged in all modes.
```
If S=1, Rd=R15; should not be used in user mode:
```
  CPSR = SPSR_<current mode>
  PC = result
  For example: MOVS PC,R14  ;return from SWI (PC=R14_svc, CPSR=SPSR_svc).
```
If S=0: Flags are not affected (not allowed for CMP,CMN,TEQ,TST).

The instruction "MOV R0,R0" is used as "NOP" opcode in 32bit ARM state.
Execution Time: (1+p)S+rI+pN. Whereas r=1 if I=0 and R=1 (ie. shift by register); otherwise r=0. And p=1 if Rd=R15; otherwise p=0.

ARM Opcodes: Multiply and Multiply-Accumulate (MUL, MLA)

Opcode Format
```
  Bit    Expl.
  31-28  Condition
  27-25  Must be 000b for this instruction
  24-21  Opcode
          0000b: MUL{cond}{S}   Rd,Rm,Rs        ;Rd=Rm*Rs     ;\works as both
          0001b: MLA{cond}{S}   Rd,Rm,Rs,Rn     ;Rd=Rm*Rs+Rn  ;/signed+unsigned
          0010b: UMAAL{cond}    RdLo,RdHi,Rm Rs ;RdHiLo=Rm*Rs+RdHi+RdLo;\un-
          0100b: UMULL{cond}{S} RdLo,RdHi,Rm,Rs ;RdHiLo=Rm*Rs          ; signed
          0101b: UMLAL{cond}{S} RdLo,RdHi,Rm,Rs ;RdHiLo=Rm*Rs+RdHiLo   ;/
          0110b: SMULL{cond}{S} RdLo,RdHi,Rm,Rs ;RdHiLo=Rm*Rs
          0111b: SMLAL{cond}{S} RdLo,RdHi,Rm,Rs ;RdHiLo=Rm*Rs+RdHiLo
          1000b: SMLAxy{cond}   Rd,Rm,Rs,Rn     ;Rd=HalfRm*HalfRs+Rn
          1001b: SMLAWy{cond}   Rd,Rm,Rs,Rn     ;Rd=(Rm*HalfRs)/10000h+Rn
          1001b: SMULWy{cond}   Rd,Rm,Rs        ;Rd=(Rm*HalfRs)/10000h
          1010b: SMLALxy{cond}  RdLo,RdHi,Rm,Rs ;RdHiLo=RdHiLo+HalfRm*HalfRs
          1011b: SMULxy{cond}   Rd,Rm,Rs        ;Rd=HalfRm*HalfRs
  20     S - Set Condition Codes (0=No, 1=Yes) (Must be 0 for Halfword & UMAAL)
  19-16  Rd (or RdHi) - Destination Register (R0-R14)
  15-12  Rn (or RdLo) - Accumulate Register  (R0-R14) (Set to 0000b if unused)
  11-8   Rs - Operand Register               (R0-R14)
  For Non-Halfword Multiplies
    7-4  Must be 1001b for these instructions
  For Halfword Multiplies
    7    Must be 1 for these instructions
    6    y - Rs Top/Bottom flag (0=B=Lower 16bit, 1=T=Upper 16bit)
    5    x - Rm Top/Bottom flag (as above), or 0 for SMLAW, or 1 for SMULW
    4    Must be 0 for these instructions
  3-0    Rm - Operand Register               (R0-R14)
```

Multiply and Multiply-Accumulate (MUL, MLA)
Restrictions: Rd may not be same as Rm. Rd,Rn,Rs,Rm may not be R15.
Note: Only the lower 32bit of the internal 64bit result are stored in Rd, thus no sign/zero extension is required and MUL and MLA can be used for both signed and unsigned calculations!
Execution Time: 1S+mI for MUL, and 1S+(m+1)I for MLA. Whereas 'm' depends on whether/how many most significant bits of Rs are all zero or all one. That is m=1 for Bit 31-8, m=2 for Bit 31-16, m=3 for Bit 31-24, and m=4 otherwise.

Flags (if S=1): Z=zeroflag, N=signflag, C=destroyed (ARMv4 and below) or C=not affected (ARMv5 and up), V=not affected. MUL/MLA supported by ARMv2 and up.

Multiply Long and Multiply-Accumulate Long (MULL, MLAL)
Optionally supported, INCLUDED in ARMv3M, EXCLUDED in ARMv4xM/ARMv5xM.
Restrictions: RdHi,RdLo,Rm must be different registers. R15 may not be used.
Execution Time: 1S+(m+1)I for MULL, and 1S+(m+2)I for MLAL. Whereas 'm' depends on whether/how many most significant bits of Rs are "all zero" (UMULL/UMLAL) or "all zero or all one" (SMULL,SMLAL). That is m=1 for Bit31-8, m=2 for Bit31-16, m=3 for Bit31-24, and m=4 otherwise.
Flags (if S=1): Z=zeroflag, N=signflag, C=destroyed (ARMv4 and below) or C=not affected (ARMv5 and up), V=destroyed??? (ARMv4 and below???) or V=not affected (ARMv5 and up).

Signed Halfword Multiply (SMLAxy,SMLAWy,SMLALxy,SMULxy,SMULWy)
Supported by E variants of ARMv5 and up, ie. ARMv5TE(xP).
Q-flag gets set on 32bit SMLAxy/SMLAWy addition overflows, however, the result is NOT truncated (as it'd be done with QADD opcodes).
Q-flag is NOT affected on (rare) 64bit SMLALxy addition overflows.
SMULxy/SMULWy cannot overflow, and thus leave Q-flag unchanged as well.
NZCV-flags are not affected by Halfword multiplies.
Execution Time: 1S+Interlock (SMULxy,SMLAxy,SMULWx,SMLAWx)
Execution Time: 1S+1I+Interlock (SMLALxy)

Unsigned Multiply Accumulate Accumulate Long (UMAAL)
Multiply with two 32bit additions, supposed to be used for cryptography.
Flags are not affected.
ARMv6 and above.

ARM Opcodes: Special ARM9 Instructions (CLZ, QADD/QSUB)

Count Leading Zeros (CLZ)

```
Bit    Expl.
31-28  Condition
27-16  Must be 0001.0110.1111b for this instruction
       Opcode (fixed)
         CLZ{cond} Rd,Rm  ;Rd=Number of leading zeros in Rm
15-12  Rd - Destination Register              (R0-R14)
11-4   Must be 1111.0001b for this instruction
3-0    Rm - Source Register                   (R0-R14)
```

CLZ supported by ARMv5 and up. Execution time: 1S.
Return: No Flags affected. Rd=0..32.

Opcode Format (QADD/QSUB)

```
Bit    Expl.
31-28  Condition
27-24  Must be 0001b for this instruction
23-20  Opcode
        0000b: QADD{cond}  Rd,Rm,Rn    ;Rd=Rm+Rn
        0010b: QSUB{cond}  Rd,Rm,Rn    ;Rd=Rm-Rn
        0100b: QDADD{cond} Rd,Rm,Rn    ;Rd=Rm+Rn*2 (doubled)
        0110b: QDSUB{cond} Rd,Rm,Rn    ;Rd=Rm-Rn*2 (doubled)
19-16  Rn - Second Source Register            (R0-R14)
15-12  Rd - Destination Register              (R0-R14)
11-4   Must be 00000101b for this instruction
3-0    Rm - First Source Register             (R0-R14)
```

Supported by E variants of ARMv5 and up, ie. ARMv5TE(xP).
Execution time: 1S+Interlock.

Results truncated to signed 32bit range in case of overflows, with the Q-flag being set (and being left unchanged otherwise). NZCV flags are not affected.
Note: Rn*2 is internally processed first, and may get truncated - even if the final result would fit into range.

ARM Opcodes: Special ARM11 Instructions (Misc)

Load/Store Byte/Halfword/Doubleword Exclusive (LDREX/STREX)

```
  Bit    Expl.
  31-28  Condition
  27-23  Must be 00011b for this instruction
  22-20  Opcode
           0: STREX{cond}  Rd, Rm, [Rn]  ;[Rn]=Rm, Rd=flag       ;32bit
           1: LDREX{cond}  Rd, [Rn]      ;Rd=[Rm]                ;32bit
           2: STREXD{cond} Rd, Rm, [Rn]  ;[Rn]=Rm,Rm+1, Rd=flag  ;64bit
           3: LDREXD{cond} Rd, [Rn]      ;Rd,Rd+1=[Rm]           ;64bit
           4: STREXB{cond} Rd, Rm, [Rn]  ;[Rn]=Rm, Rd=flag       ;8bit
           5: LDREXB{cond} Rd, [Rn]      ;Rd=[Rm]                ;8bit
           6: STREXH{cond} Rd, Rm, [Rn]  ;[Rn]=Rm, Rd=flag       ;16bit
           7: LDREXH{cond} Rd, [Rn]      ;Rd=[Rm]                ;16bit
  19-16  Rn - Base Register                     (R0-R14)
  15-12  Rd - Destination Register              (R0-R14)
  11-4   Must be 1111.1001b for this instruction
  3-0    Rm - Source Register (1111b for LDREX)  (R0-R14)
```

For exclusive access to shared memory regions (ie. in dual-core CPUs?)
Unknown what "exclusive" means?
Return: No Flags affected.
ARMv6 and up supports LDREX/STREX (with 32bit operand).
ARMv6K and up supports LDREX/STREX{B|H|D} (with 8/16/64bit operand).

Clear Exclusive (CLREX)

```
  Bit    Expl.
  31-0   Opcode 1111.0101.0111.1111.1111.0000.0001.1111b for CLREX
```

Supported on ARMv6K and up.

Hint (TrueNOP, YIELD, WFE, WFI, SEV)

```
  Bit    Expl.
  31-28  Condition (should be always 0Eh=Always for NOP)
  27-8   Must be 0011.0010.0000.1111.0000b
  7-0    Opcode/Hint
           00h: TrueNOP      ;no operation
           01h: YIELD{cond}
           02h: WFE{cond}    ;wait for event
           03h: WFI{cond}    ;wait for interrupt
           04h: SEV{cond}    ;send event
```

Supported on ARMv6K and up.

Change Processor State aka CPSR Mode/Flags (CPS, CPSID, CPSIE)

```
  31-20  Must be 1111.0001.0000b
  19-17  Opcode
          001b: CPS   #<mode>           ;change mode
          100b: CPSIE {A}{I}{F}         ;interrupt bit(s) enable
          101b: CPSIE {A}{I}{F},#<mode> ;interrupt bit(s) enable & change mode
          110b: CPSID {A}{I}{F}         ;interrupt bit(s) disable
          111b: CPSID {A}{I}{F},#<mode> ;interrupt bit(s) disable & change mode
  16-9   Must be 0000.0000b
  8      Change A flag (what?) (0=No, 1=Yes)  ;\must be all zero for CPS,
  7      Change I flag         (0=No, 1=Yes)  ; must be at least one flag set
  6      Change F flag         (0=No, 1=Yes)  ;/for CPSID/CPSIE
```

```
  5      Must be 0
  4-0    Mode (00h..1Fh) (must be 00h if bit17=0)
```

Set Endianness for Data Access (SETEND)

```
  31-10  Fixed, 11110001000000010000000b
  9      Opcode/operand
          0: SETEND LE  ;clears the CPSR E bit
          1: SETEND BE  ;sets the CPSR E bit
  8-0    Fixed, 000000000b
```

Changes the byte order for data access to Big/Little endian. Does probably not affect opcode fetches, but probably affect all other memory including push/pop? Or affects "LDR" only?
Supported on ARMv6 and above.

Store Return State (SRS) (stores the R14 and SPSR of the current mode)

to the word at the specified address and the following word respectively. The address is determined from the banked version of R13 belonging to a specified mode.

```
  Syntax
  SRS<addressing_mode> #<mode>{!}   ;<-- original (pre-UAL)
  SRS<addressing_mode> sp!,#<mode>  ;<-- more common (UAL)
```

<addressing_mode>: Is similar to the <addressing_mode> in LDM and STM instructions, see Addressing Mode 4 - Load and Store Multiple on page A5-41, but with the following differences:
- The base register, Rn, is the banked version of R13 for the mode specified by <mode>, rather than the current mode.
- The number of registers to store is 2.
- The register list is {R14, SPSR}, with both R14 and the SPSR being the versions belonging to the current mode.

<mode> Specifies the number of the mode whose banked register is used as the base register for <addressing_mode>. The mode number is the 5-bit encoding of the chosen mode in a PSR, as described in The mode bits on page A2-14.
! If present, sets the W bit. This causes the instruction to write a modified value back to its base register, in a manner similar to that specified for Addressing Mode 4 - Load and Store Multiple on page A5-41. If ! is omitted, the W bit is 0 and the instruction does not change the base register.

```
  31-25  Fixed, 1111100b
  24     P
  23     U
  22     Fixed, 1
  21     W
  20     Fixed, 0 (store)
  19-5   Fixed, 1101000001010000b
  4-0    mode
```

Supported on ARMv6 and above.

Return From Exception (RFE) (RFEDA, RFEDB, RFEIA, RFEIB)

loads the PC and the CPSR from the word at the specified address and the
following word respectively.

```
  Syntax
  RFE<addressing_mode> Rn{!}
```

<addressing_mode>: Is similar to the <addressing_mode> in LDM and STM instructions, see Addressing Mode 4 - Load and Store Multiple on page A5-41, but with the following differences:
- The number of registers to load is 2.
- The register list is {PC, CPSR}.

<Rn> Specifies the base register to be used by <addressing_mode>. If R15 is specified as the base register, the result is UNPREDICTABLE.
! If present, sets the W bit. This causes the instruction to write a modified value back to its base register, in a manner similar to that specified for Addressing Mode 4 - Load and Store Multiple on page A5-41. If ! is omitted, the W bit is 0 and the instruction does not change the base register.
While RFE supports different base registers, a general usage case is where Rn == sp (the stack pointer), held in

R13. The instruction can then be used as the return method associated with instructions SRS and CPS. See New instructions to improve exception handling on page A2-28 for more details.

```
  31-25  Fixed, 1111100b
  24     P
  23     U
  22     Fixed, 0
  21     W
  20     Fixed, 1 (load)
  19-16  Rn
  15-0   Fixed, 0000101000000000b
```

Supported on ARMv6 and above.

SVC (formerly SWI)

SWI was renamed to SVC in new UAL syntax.

CPY

CPY Rd,Rm is an alias for MOV Rd,Rm (without flags affected).

```
  31-28  Condition
  27-16  Fixed, 000110100000
  15-12  Rd - Destination Register
  11-4   Fixed, 00000000
  3-0    Rm
```

The alias was invented in ARMv6 for THUMB and ARM code (apparently because pre-UAL THUMB syntax couldn't distinguish between MOV and MOVS) (ie. in ARM it's useless, but in THUMB it does allow using a formerly undocumented opcode).
ARMv6 and above.

ARM Opcodes: Special ARM11 Instructions (SIMD)

Sign Extend / Zero Extend, with optional Addition (SXT, UXT)

```
  31-28  Condition
  27-23  Fixed, 01101b
  22-20  Opcode:
          0: SXTAB16{cond} Rd, Rn, Rm{,ROR #imm} ;\
          2: SXTAB{cond}   Rd, Rn, Rm{,ROR #imm} ;
          3: SXTAH{cond}   Rd, Rn, Rm{,ROR #imm} ; with add
          4: UXTAB16{cond} Rd, Rn, Rm{,ROR #imm} ;
          6: UXTAB{cond}   Rd, Rn, Rm{,ROR #imm} ;
          7: UXTAH{cond}   Rd, Rn, Rm{,ROR #imm} ;/
          0: SXTB16{cond}  Rd,     Rm{,ROR #imm} ;\
          2: SXTB{cond}    Rd,     Rm{,ROR #imm} ;
          3: SXTH{cond}    Rd,     Rm{,ROR #imm} ; without add
          4: UXTB16{cond}  Rd,     Rm{,ROR #imm} ;
          6: UXTB{cond}    Rd,     Rm{,ROR #imm} ;
          7: UXTH{cond}    Rd,     Rm{,ROR #imm} ;/
  19-16  Rn - Addition Register (or 1111b when no addition)          (R0-R14)
  15-12  Rd - Destination Register                                   (R0-R14)
  11-10  Rotation (0=None/ROR#0, 1=ROR#8, 2=ROR#16, 3=ROR#24)
  9-4    Fixed, 000111b
  3-0    Rm - Source register (to be rotated/extracted from)         (R0-R14)
```

Supported on ARMv6 and above.

Select Bytes from 1st/2nd operand, depending on GE flags (SEL)

Syntax

```
  SEL{cond} Rd, Rn, Rm
```

Operation

if ConditionPassed(cond) then
Rd[7:0] = if GE[0] == 1 then Rn[7:0] else Rm[7:0]

Rd[15:8] = if GE[1] == 1 then Rn[15:8] else Rm[15:8]
Rd[23:16] = if GE[2] == 1 then Rn[23:16] else Rm[23:16]
Rd[31:24] = if GE[3] == 1 then Rn[31:24] else Rm[31:24]
Usage
Use SEL after instructions such as SADD8, SADD16, SSUB8, SSUB16, UADD8, UADD16, USUB8, USUB16, SADDSUBX, SSUBADDX, UADDSUBX and USUBADDX, that set the GE flags. For example, the following sequence of instructions sets each byte of Rd equal to the unsigned minimum of the corresponding bytes of Ra and Rb:

```
USUB8 Rd, Ra, Rb
SEL Rd, Rb, Ra
```

Notes
Specifying R15 for register Rd, Rm, or Rn has UNPREDICTABLE results.

```
31-28  Condition
27-20  Fixed, 01101000b
19-16  Rn - First Source Register (8bit snippets used when GE bits = 1)
15-12  Rd - Destination Register
11-4   Fixed, 11111011b
3-0    Rm - Second Source Register (8bit snippets used when GE bits = 0)
```

ARMv6 and above.

SADD16
SADD16 (Signed Add) performs two 16-bit signed integer additions. It sets the GE bits in the CPSR according to the results of the additions.
ARMv6 and above.
Operation
sum = Rn[15:0] + Rm[15:0] /* Signed addition */
Rd[15:0] = sum[15:0]
GE[1:0] = if sum >= 0 then 0b11 else 0
sum = Rn[31:16] + Rm[31:16] /* Signed addition */
Rd[31:16] = sum[15:0]
GE[3:2] = if sum >= 0 then 0b11 else 0

```
31-28  Condition
27-23  Fixed, 01100b
22-20  Opcode, part 1:
        001b: for Sxx           (signed, with GE flags)
        010b: for Qxx           (signed, with saturation)
        011b: for SHxx          (signed, with halving)
        101b: for Uxx           (unsigned, with GE flags)
        110b: for UQxx          (unsigned, with saturation)
        111b: for UHxx          (unsigned, with halving)
19-16  Rn - First Source Register                                  (R0-R14)
15-12  Rd - Destination Register                                   (R0-R14)
11-8   Fixed, 1111b
7-5    Opcode, part 2:
        000b: xxADD16{cond}   Rd, Rn, Rm
        001b: xxADDSUBX{cond} Rd, Rn, Rm
        010b: xxSUBADDX{cond} Rd, Rn, Rm
        011b: xxSUB16{cond}   Rd, Rn, Rm
        100b: xxADD8{cond}    Rd, Rn, Rm
        111b: xxSUB8{cond}    Rd, Rn, Rm
4      Fixed, 1
3-0    Rm - Second Source register                                 (R0-R14)
```

SMLAD
SMLAD (Signed Multiply Accumulate Dual) performs two signed 16 x 16-bit multiplications. It adds the products to a 32-bit accumulate operand.
Optionally, you can exchange the halfwords of the second operand before performing the arithmetic. This produces top x bottom and bottom x top multiplication.
This instruction sets the Q flag if the accumulate operation overflows. Overflow cannot occur during the

multiplications.
X Sets the X bit of the instruction to 1, and the multiplications are bottom x top and top x bottom.
If the X is omitted, sets the X bit to 0, and the multiplications are bottom x bottom and top x top.
<Rm> Specifies the register that contains the first operand.
<Rs> Specifies the register that contains the second operand.
<Rn> Specifies the register that contains the accumulate operand.

```
  31-28  Condition
  27-23  Fixed, 01110
  22-20  Opcode part 1 (part2 in bit7-6)
         Merged part1.part2:
           000.00: SMUAD{X}{cond}  Rd,Rm,Rs
           000.00: SMLAD{X}{cond}  Rd,Rm,Rs,Rn
           000.01: SMUSD{X}{cond}  Rd,Rm,Rs
           000.01: SMLSD{X}{cond}  Rd,Rm,Rs,Rn
           100.00: SMLALD{X}{cond} RdLo,RdHi,Rm,Rs
           100.01: SMLSLD{X}{cond} RdLo,RdHi,Rm,Rs
           101.00: SMMUL{R}{cond}  Rd,Rm,Rs        ;Rd=(Rm*Rs)/100000000h
           101.00: SMMLA{R}{cond}  Rd,Rm,Rs,Rn     ;Rd=(Rm*Rs)/100000000h+Rz
           101.11: SMMLS{R}{cond}  Rd,Rm,Rs,Rn     ;Rd=(Rm*Rs)/100000000h-Rz
  19-16  Rd or RdHi - Destination Register
  15-12  Rn or RdLo or unused (must be 0Fh for SMUAD/SMUSD/SMMUL)
  11-8   Rs
  7-6    Opcode part 2 (part1 in bit22-20, see there)
  5      X or R flag   (X=Exchange, R=Rounded)
  4      Fixed, 1
  3-0    Rm
```

ARMv6 and above.

Most significant word multiply

There are three signed 32-bit x 32-bit Multiply instructions that produce top 32-bit results:

```
  SMMUL Multiplies the 32-bit values of two registers together, and stores
        the top 32 bits of the signed 64-bit result in a third register.
  SMMLA Multiplies the 32-bit values of two registers together, extracts the
        top 32 bits, adds the 32-bit value from a third register, and stores
        the signed 32-bit result in a fourth register.
  SMMLS Multiplies the 32-bit value of two registers together, extracts the
        top 32 bits, subtracts this from a 32-bit value from a third
        register, and stores the signed 32-bit result in a fourth register.
```

The above three instructions do not affect any flags.

Dual halfword multiply

There are six dual, signed 16-bit x 16-bit Multiply instructions:

```
  SMUAD Multiplies the values of the top halfwords of two registers together,
        multiplies the values of the bottom halfwords of the same two
        registers together, adds the products, and stores the 32-bit result
        in a third register.
  SMUSD Multiplies the values of the top halfwords of two registers together,
        multiplies the values of the bottom halfwords of the same two
        registers together, subtracts one product from the other, and stores
        the 32-bit result in a third register.
  ---(descriptions below seem to be bugged, seem to describe 32bit SMMxx)---
  SMLAD Multiplies the 32-bit value of two registers together, extracts the
        top 32 bits, subtracts this from a 32-bit value from a third
        register, and stores the signed 32-bit result in a fourth register.
  SMLSD Multiplies the 32-bit values of two registers together, extracts the
        top 32 bits, adds the 32-bit value from a third register, and stores
        the signed 32-bit result in a fourth register.
  SMLALD Multiplies the 32-bit value of two registers together, extracts the
         top 32 bits, subtracts this from a 32-bit value from a third
         register, and stores the signed 32-bit result in a fourth register.
  SMLSLD Multiplies the 32-bit value of two registers together, extracts the
         top 32 bits, subtracts this from a 32-bit value from a third
         register, and stores the signed 32-bit result in a fourth register.
```

SMUAD, SMLAD, and SMLSLD can set the Q flag if overflow occurs in the operation. All other instructions do not affect any flags.

Signed/Unsigned Saturate (SSAT, USAT)
Saturates a 32bit value (with optional shift) to the specified range (of 1..32 bits).

```
  31-28  Condition
  27-21  Fixed, 0110101b: SSAT{cond} Rd, #<range>, Rm {,LSL/ASR #imm}
                0110111b: USAT{cond} Rd, #<range>, Rm {,LSL/ASR #imm}
  20-16  Immediate: range-1 (0..31 = Saturate to 1..32 bits range)
  15-12  Rd - Destination Register
  11-7   shift_imm (1..31 for LSL/ASR#1..31, or 0 for LSL#0, or 0 for ASR#32)
  6      Shift type (0=LSL, 1=ASR) (note: really ASR, even "unsigned" USAT)
  5-4    Fixed, 01b
  3-0    Rm - Source Register
```

The Q flag is set if the operation saturates.
ARMv6 and above.

Signed/Unsigned Saturate on Halfwords (SSAT16, USAT16)
Saturates two 16bit values to the specified range (of 1..16 bits).

```
  31-28  Condition
  27-21  Fixed, 0110101b: SSAT16{cond} Rd, #<range>, Rm
                0110111b: USAT16{cond} Rd, #<range>, Rm
  20     Fixed, 0
  19-16  Immediate: range-1 (0..15 = Saturate to 1..16 bits range)
  15-12  Rd - Destination Register
  11-4   Fixed, 11110011b
  3-0    Rm - Source Register
```

The Q flag is set if either halfword operation saturates.
ARMv6 and above.

Unsigned Sum of Absolute Differences, optional Accumulate (USAD8, USADA8)
Performs four unsigned 8bit subtractions, and adds the absolute values of the differences together.

```
  31-28  Condition
  27-20  Fixed, 01111000b
  19-16  Rd - Destination Register
  15-12  Rn (if any) (otherwise fixed, 1111)
           USAD8{cond}  Rd, Rm, Rs        ;without Rn   ;Rd=sum
           USADA8{cond} Rd, Rm, Rs, Rn    ;with Rn      ;Rd=sum + Rn
  11-8   Rs
  7-4    Fixed, 0001b
  3-0    Rm
```

ARMv6 and above.

Pack Halfword, Bottom-Top or Top-Bottom (PKHBT, PKHTB)
PKHBT (Pack Halfword Bottom Top) combines the bottom (least significant) halfword of its first operand with the top (most significant) halfword of its shifted second operand. The shift is a left shift, by any amount from 0 to 31.
<Rn> Specifies the register that contains the first operand. Bits[15:0] of this operand become bits[15:0] of the result of the operation.
<Rm> Specifies the register that contains the second operand. This is shifted left by the specified amount, then bits[31:16] of this operand become bits[31:16] of the result of the operation.
<shift_imm> Specifies the amount by which Rm is to be shifted left. This is a value from 0 to 31.
If the shift specifier is omitted, a left shift by 0 is used.
Operation
if ConditionPassed(cond) then
Rd[15:0] = Rn[15:0]
Rd[31:16] = (Rm Logical_Shift_Left shift_imm)[31:16]

```
  31-28  Condition
  27-20  Fixed, 01101000b
```

```
  19-16  Rn
  15-12  Rd - Destination Register
  11-7   shift_imm (0..31)
  6-4    Fixed, 001  PKHBT {cond} Rd, Rn, Rm {, LSL #<shift_imm>}
                101  PKHTB {cond} Rd, Rn, Rm {, ASR #<shift_imm>}
  3-0    Rm
```
ARMv6 and above.

Reverse Byte Order for Big/Little-Endian Conversion (REV, REV16, REVSH)
Specifying R15 for register Rd or Rm has UNPREDICTABLE results.
```
  31-28  Condition
  27-23  Fixed, 01101b
  22     Opcode (with bit7)
          Bit22=0, Bit7=0: REV{cond}   Rd, Rm   ;reverse 32bit
          Bit22=0, Bit7=1: REV16{cond} Rd, Rm
          Bit22=1, Bit7=1: REVSH{cond} Rd, Rm
  21-16  Fixed, 111111b
  15-12  Rd - Destination Register
  11-8   Fixed, 1111b
  7      Opcode (with bit22, see there)
  6-4    Fixed, 011b
  3-0    Rm - Source Register
```
ARMv6 and above.

ARM Opcodes: PSR Transfer (MRS, MSR)

Opcode Format
These instructions occupy an unused area (TEQ,TST,CMP,CMN with S=0) of ALU opcodes.
```
  Bit    Expl.
  31-28  Condition
  27-26  Must be 00b for this instruction
  25     I - Immediate Operand Flag  (0=Register, 1=Immediate) (Zero for MRS)
  24-23  Must be 10b for this instruction
  22     Psr - Source/Destination PSR  (0=CPSR, 1=SPSR_<current mode>)
  21     Opcode
           0: MRS{cond} Rd,Psr          ;Rd = Psr
           1: MSR{cond} Psr{_field},Op  ;Psr[field] = Op
  20     Must be 0b for this instruction (otherwise TST,TEQ,CMP,CMN)
  For MRS:
    19-16   Must be 1111b for this instruction (otherwise SWP)
    15-12   Rd - Destination Register  (R0-R14)
    11-0    Not used, must be zero.
  For MSR:
    19      f  write to flags field     Bit 31-24 (aka _flg)
    18      s  write to status field    Bit 23-16 (reserved, don't change)
    17      x  write to extension field Bit 15-8  (reserved, don't change)
    16      c  write to control field   Bit 7-0   (aka _ctl)
    15-12   Not used, must be 1111b.
  For MSR Psr,Rm (I=0)
    11-4    Not used, must be zero. (otherwise BX)
    3-0     Rm - Source Register <op>  (R0-R14)
  For MSR Psr,Imm (I=1)
    11-8    Shift applied to Imm   (ROR in steps of two 0-30)
    7-0     Imm - Unsigned 8bit Immediate
    In source code, a 32bit immediate should be specified as operand.
    The assembler should then convert that into a shifted 8bit value.
```
MSR/MRS and CPSR/SPSR supported by ARMv3 and up.
ARMv2 and below contained PSR flags in R15, accessed by CMP/CMN/TST/TEQ{P}.
The field mask bits specify which bits of the destination Psr are write-able (or write-protected), one or more of these bits should be set, for example, CPSR_fsxc (aka CPSR aka CPSR_all) unlocks all bits (see below user

mode restriction though).
Restrictions:
In non-privileged mode (user mode): only condition code bits of CPSR can be changed, control bits can't.
Only the SPSR of the current mode can be accessed; In User and System modes no SPSR exists.
The T-bit may not be changed; for THUMB/ARM switching use BX instruction.
Unused Bits in CPSR are reserved for future use and should never be changed (except for unused bits in the flags field).
Execution Time: 1S.

Note: The A22i assembler recognizes MOV as alias for both MSR and MRS because it is practically not possible to remember whether MSR or MRS was the load or store opcode, and/or whether it does load to or from the Psr register.

ARM Opcodes: Memory: Single Data Transfer (LDR, STR, PLD)

Opcode Format

```
  Bit    Expl.
  31-28  Condition (Must be 1111b for PLD)
  27-26  Must be 01b for this instruction
  25     I - Immediate Offset Flag (0=Immediate, 1=Shifted Register)
  24     P - Pre/Post (0=post; add offset after transfer, 1=pre; before trans.)
  23     U - Up/Down Bit (0=down; subtract offset from base, 1=up; add to base)
  22     B - Byte/Word bit (0=transfer 32bit/word, 1=transfer 8bit/byte)
  When above Bit 24 P=0 (Post-indexing, write-back is ALWAYS enabled):
    21     T - Memory Management (0=Normal, 1=Force non-privileged access)
  When above Bit 24 P=1 (Pre-indexing, write-back is optional):
    21     W - Write-back bit (0=no write-back, 1=write address into base)
  20     L - Load/Store bit (0=Store to memory, 1=Load from memory)
          0: STR{cond}{B}{T} Rd,<Address>   ;[Rn+/-<offset>]=Rd
          1: LDR{cond}{B}{T} Rd,<Address>   ;Rd=[Rn+/-<offset>]
         (1: PLD <Address> ;Prepare Cache for Load, see notes below)
          Whereas, B=Byte, T=Force User Mode (only for POST-Indexing)
  19-16  Rn - Base register               (R0..R15) (including R15=PC+8)
  15-12  Rd - Source/Destination Register (R0..R15) (including R15=PC+12)
  When above I=0 (Immediate as Offset)
    11-0   Unsigned 12bit Immediate Offset (0-4095, steps of 1)
  When above I=1 (Register shifted by Immediate as Offset)
    11-7   Is - Shift amount      (1-31, 0=Special/See below)
    6-5    Shift Type             (0=LSL, 1=LSR, 2=ASR, 3=ROR)
    4      Must be 0 (Reserved, see The Undefined Instruction)
    3-0    Rm - Offset Register   (R0..R14) (not including PC=R15)
```

Instruction Formats for <Address>

An expression which generates an address:

```
  <expression>                ;an immediate used as address
  ;*** restriction: must be located in range PC+/-4095+8, if so,
  ;*** assembler will calculate offset and use PC (R15) as base.
```

Pre-indexed addressing specification:

```
  [Rn]                        ;offset = zero
  [Rn, <#{+/-}expression>]{!}   ;offset = immediate
  [Rn, {+/-}Rm{,<shift>} ]{!}   ;offset = register shifted by immediate
```

Post-indexed addressing specification:

```
  [Rn], <#{+/-}expression>      ;offset = immediate
  [Rn], {+/-}Rm{,<shift>}       ;offset = register shifted by immediate
```

Whereas...

```
  <shift>  immediate shift such like LSL#4, ROR#2, etc. (see ALU opcodes).
  {!}      exclamation mark ("!") indicates write-back (Rn will be updated).
```

Notes
Shift amount 0 has special meaning, as described for ALU opcodes.
When writing a word (32bit) to memory, the address should be word-aligned.
When reading a byte from memory, upper 24 bits of Rd are zero-extended.
LDR PC,<op> on ARMv4 leaves CPSR.T unchanged.
LDR PC,<op> on ARMv5 sets CPSR.T to <op> Bit0, (1=Switch to Thumb).

When reading a word from a halfword-aligned address (which is located in the middle between two word-aligned addresses), the lower 16bit of Rd will contain [address] ie. the addressed halfword, and the upper 16bit of Rd will contain [Rd-2] ie. more or less unwanted garbage. However, by isolating lower bits this may be used to read a halfword from memory. (Above applies to little endian mode, as used in GBA.)

In a virtual memory based environment (ie. not in the GBA), aborts (ie. page faults) may take place during execution, if so, Rm and Rn should not specify the same register when post-indexing is used, as the abort-handler might have problems to reconstruct the original value of the register.

Return: CPSR flags are not affected.
Execution Time: For normal LDR: 1S+1N+1I. For LDR PC: 2S+2N+1I. For STR: 2N.

PLD <Address> ;Prepare Cache for Load
PLD must use following settings cond=1111b, P=1, B=1, W=0, L=1, Rd=1111b, the address may not use post-indexing, and may not use writeback, the opcode is encoded identical as LDRNVB R15,<Address>.
PLD signalizes to the memory system that a specific memory address will be soon accessed, the memory system may use this hint to prepare caching/pipelining, aside from that, PLD does not have any affect to the program logic, and behaves identical as NOP.
PLD supported by ARMv5TE and up only, not ARMv5, not ARMv5TExP.

ARM Opcodes: Memory: Halfword, Doubleword, and Signed Data Transfer

Opcode Format

```
  Bit    Expl.
  31-28  Condition
  27-25  Must be 000b for this instruction
  24     P - Pre/Post (0=post; add offset after transfer, 1=pre; before trans.)
  23     U - Up/Down Bit (0=down; subtract offset from base, 1=up; add to base)
  22     I - Immediate Offset Flag (0=Register Offset, 1=Immediate Offset)
  When above Bit 24 P=0 (Post-indexing, write-back is ALWAYS enabled):
    21     Not used, must be zero (0)
  When above Bit 24 P=1 (Pre-indexing, write-back is optional):
    21     W - Write-back bit (0=no write-back, 1=write address into base)
  20     L - Load/Store bit (0=Store to memory, 1=Load from memory)
  19-16  Rn - Base register               (R0-R15) (Including R15=PC+8)
  15-12  Rd - Source/Destination Register  (R0-R15) (Including R15=PC+12)
  11-8   When above Bit 22 I=0 (Register as Offset):
           Not used. Must be 0000b
         When above Bit 22 I=1 (immediate as Offset):
           Immediate Offset (upper 4bits)
  7      Reserved, must be set (1)
  6-5    Opcode (0-3)
         When Bit 20 L=0 (Store) (and Doubleword Load/Store):
          0: Reserved for SWP instruction
          1: STR{cond}H  Rd,<Address>  ;Store halfword   [a]=Rd
          2: LDR{cond}D  Rd,<Address>  ;Load Doubleword  R(d)=[a], R(d+1)=[a+4]
          3: STR{cond}D  Rd,<Address>  ;Store Doubleword [a]=R(d), [a+4]=R(d+1)
```

```
        When Bit 20 L=1 (Load):
         0: Reserved.
         1: LDR{cond}H  Rd,<Address>  ;Load Unsigned halfword (zero-extended)
         2: LDR{cond}SB Rd,<Address>  ;Load Signed byte (sign extended)
         3: LDR{cond}SH Rd,<Address>  ;Load Signed halfword (sign extended)
  4     Reserved, must be set (1)
  3-0   When above Bit 22 I=0:
          Rm - Offset Register            (R0-R14) (not including R15)
        When above Bit 22 I=1:
          Immediate Offset (lower 4bits)  (0-255, together with upper bits)
```

STRH,LDRH,LDRSB,LDRSH supported on ARMv4 and up.
STRD/LDRD supported on ARMv5TE and up only, not ARMv5, not ARMv5TExP.
STRD/LDRD: base writeback: Rn should not be same as R(d) or R(d+1).
STRD: index register: Rm should not be same as R(d) or R(d+1).
STRD/LDRD: Rd must be an even numbered register (R0,R2,R4,R6,R8,R10,R12).
STRD/LDRD: Address must be double-word aligned (multiple of eight).

Instruction Formats for <Address>
An expression which generates an address:

```
  <expression>                  ;an immediate used as address
  ;*** restriction: must be located in range PC+/-255+8, if so,
  ;*** assembler will calculate offset and use PC (R15) as base.
```

Pre-indexed addressing specification:

```
  [Rn]                          ;offset = zero
  [Rn, <#{+/-}expression>]{!}   ;offset = immediate
  [Rn, {+/-}Rm]{!}              ;offset = register
```

Post-indexed addressing specification:

```
  [Rn], <#{+/-}expression>      ;offset = immediate
  [Rn], {+/-}Rm                 ;offset = register
```

Whereas...

```
  {!}      exclamation mark ("!") indicates write-back (Rn will be updated).
```

Return: No Flags affected.
Execution Time: For Normal LDR, 1S+1N+1I. For LDR PC, 2S+2N+1I. For STRH 2N.

ARM Opcodes: Memory: Block Data Transfer (LDM, STM)

Opcode Format

```
  Bit    Expl.
  31-28  Condition
  27-25  Must be 100b for this instruction
  24     P - Pre/Post (0=post; add offset after transfer, 1=pre; before trans.)
  23     U - Up/Down Bit (0=down; subtract offset from base, 1=up; add to base)
  22     S - PSR & force user bit (0=No, 1=load PSR or force user mode)
  21     W - Write-back bit (0=no write-back, 1=write address into base)
  20     L - Load/Store bit (0=Store to memory, 1=Load from memory)
          0: STM{cond}{amod} Rn{!},<Rlist>{^}  ;Store (Push)
          1: LDM{cond}{amod} Rn{!},<Rlist>{^}  ;Load  (Pop)
          Whereas, {!}=Write-Back (W), and {^}=PSR/User Mode (S)
  19-16  Rn - Base register               (R0-R14) (not including R15)
  15-0   Rlist - Register List
  (Above 'offset' is meant to be the number of words specified in Rlist.)
```

Return: No Flags affected.
Execution Time: For normal LDM, nS+1N+1I. For LDM PC, (n+1)S+2N+1I. For STM (n-1)S+2N. Where n is the number of words transferred.

Addressing Modes {amod}
The IB,IA,DB,DA suffixes directly specify the desired U and P bits:

```
  IB  increment before          ;P=1, U=1
  IA  increment after           ;P=0, U=1
  DB  decrement before          ;P=1, U=0
  DA  decrement after           ;P=0, U=0
```
Alternately, FD,ED,FA,EA could be used, mostly to simplify mnemonics for stack transfers.
```
  ED  empty stack, descending   ;LDM: P=1, U=1  ;STM: P=0, U=0
  FD  full stack,  descending   ;     P=0, U=1  ;     P=1, U=0
  EA  empty stack, ascending    ;     P=1, U=0  ;     P=0, U=1
  FA  full stack,  ascending    ;     P=0, U=0  ;     P=1, U=1
```
Ie. the following expressions are aliases for each other:
```
  STMFD=STMDB=PUSH   STMED=STMDA   STMFA=STMIB   STMEA=STMIA
  LDMFD=LDMIA=POP    LDMED=LDMIB   LDMFA=LDMDA   LDMEA=LDMDB
```
Note: The equivalent THUMB functions use fixed organization:
```
  PUSH/POP: full descending     ;base register SP (R13)
  LDM/STM:  increment after     ;base register R0..R7
```
Descending is common stack organization as used in 80x86 and Z80 CPUs, SP is decremented when pushing/storing data, and incremented when popping/loading data.

When S Bit is set (S=1)
If instruction is LDM and R15 is in the list: (Mode Changes)
```
  While R15 loaded, additionally: CPSR=SPSR_<current mode>
```
Otherwise: (User bank transfer)
```
  Rlist is referring to User Bank Registers, R0-R15 (rather than
  register related to the current mode, such like R14_svc etc.)
  Base write-back should not be used for User bank transfer.
  Caution - When instruction is LDM:
  If the following instruction reads from a banked register (eg. R14_svc),
  then CPU might still read R14 instead; if necessary insert a dummy NOP.
```

Notes
The base address should be usually word-aligned.
LDM Rn,...,PC on ARMv4 leaves CPSR.T unchanged.
LDR Rn,...,PC on ARMv5 sets CPSR.T to <op> Bit0, (1=Switch to Thumb).

Transfer Order
The lowest Register in Rlist (R0 if its in the list) will be loaded/stored to/from the lowest memory address. Internally, the rlist register are always processed with INCREASING addresses (ie. for DECREASING addressing modes, the CPU does first calculate the lowest address, and does then process rlist with increasing addresses; this detail can be important when accessing memory mapped I/O ports).

Strange Effects on Invalid Rlist's
Empty Rlist: R15 loaded/stored (ARMv4 only), and Rb=Rb+/-40h (ARMv4-v5).
Writeback with Rb included in Rlist: Store OLD base if Rb is FIRST entry in Rlist, otherwise store NEW base (STM/ARMv4), always store OLD base (STM/ARMv5), no writeback (LDM/ARMv4), writeback if Rb is "the ONLY register, or NOT the LAST register" in Rlist (LDM/ARMv5).

ARM Opcodes: Memory: Single Data Swap (SWP)

Opcode Format
```
  Bit    Expl.
  31-28  Condition
  27-23  Must be 00010b for this instruction
         Opcode (fixed)
           SWP{cond}{B} Rd,Rm,[Rn]      ;Rd=[Rn], [Rn]=Rm
  22     B - Byte/Word bit (0=swap 32bit/word, 1=swap 8bit/byte)
  21-20  Must be 00b for this instruction
  19-16  Rn - Base register                    (R0-R14)
```

```
  15-12  Rd - Destination Register              (R0-R14)
  11-4   Must be 00001001b for this instruction
  3-0    Rm - Source Register                   (R0-R14)
```
SWP/SWPB supported by ARMv2a and up. SWP/SWPB has been deprecated in ARMv6, made optional in ARMv7 (with the possibility of disabling it if still available), and removed in ARMv8.
Swap works properly including if Rm and Rn specify the same register.
R15 may not be used for either Rn,Rd,Rm. (Rn=R15 would be MRS opcode).
Upper bits of Rd are zero-expanded when using Byte quantity. For info about byte and word data memory addressing, read LDR and STR opcode description.
Execution Time: 1S+2N+1I. That is, 2N data cycles, 1S code cycle, plus 1I.

ARM Opcodes: Coprocessor Instructions (MRC/MCR, LDC/STC, CDP, MCRR/MRRC)

Coprocessor Register Transfers (MRC, MCR) (with ARM Register read/write)

```
  Bit    Expl.
  31-28  Condition (or 1111b for MRC2/MCR2 opcodes on ARMv5 and up)
  27-24  Must be 1110b for this instruction
  23-21  CP Opc - Coprocessor operation code         (0-7)
  20     ARM-Opcode (0-1)
          0: MCR{cond} Pn,<cpopc>,Rd,Cn,Cm{,<cp>}   ;move from ARM to CoPro
          0: MCR2      Pn,<cpopc>,Rd,Cn,Cm{,<cp>}   ;move from ARM to CoPro
          1: MRC{cond} Pn,<cpopc>,Rd,Cn,Cm{,<cp>}   ;move from CoPro to ARM
          1: MRC2      Pn,<cpopc>,Rd,Cn,Cm{,<cp>}   ;move from CoPro to ARM
  19-16  Cn     - Coprocessor source/dest. Register  (C0-C15)
  15-12  Rd     - ARM source/destination Register    (R0-R15)
  11-8   Pn     - Coprocessor number                 (P0-P15)
  7-5    CP     - Coprocessor information            (0-7)
  4      Reserved, must be one (1) (otherwise CDP opcode)
  3-0    Cm     - Coprocessor operand Register       (C0-C15)
```
MCR/MRC supported by ARMv2 and up, MCR2/MRC2 by ARMv5 and up.
A22i syntax allows to use MOV with Rd specified as first (dest), or last (source) operand. Native MCR/MRC syntax uses Rd as middle operand, <cp> can be ommited if <cp> is zero.
When using MCR with R15: Coprocessor will receive a data value of PC+12.
When using MRC with R15: Bit 31-28 of data are copied to Bit 31-28 of CPSR (ie. N,Z,C,V flags), other data bits are ignored, CPSR Bit 27-0 are not affected, R15 (PC) is not affected.
Execution time: 1S+bI+1C for MCR, 1S+(b+1)I+1C for MRC.
Return: For MRC only: Either R0-R14 modified, or flags affected (see above).
For details refer to original ARM docs. The opcodes are irrelevant for GBA/NDS7 because no coprocessor exists (except for a dummy CP14 unit). However, NDS9 includes a working CP15 unit. And, 3DS ARM11 uses CP10/CP11 as VFP floating point unit.
ARM CP14 ICEbreaker Debug Communications Channel
ARM CP15 System Control Coprocessor
ARM Vector Floating-point Unit (VFP)

Coprocessor Data Transfers (LDC, STC) (with Memory read/write)

```
  Bit    Expl.
  31-28  Condition (or 1111b for LDC2/STC2 opcodes on ARMv5 and up)
  27-25  Must be 110b for this instruction
  24     P - Pre/Post (0=post; add offset after transfer, 1=pre; before trans.)
  23     U - Up/Down Bit (0=down; subtract offset from base, 1=up; add to base)
  22     N - Transfer length (0-1, interpretation depends on co-processor)
  21     W - Write-back bit (0=no write-back, 1=write address into base)
  20     Opcode (0-1)
          0: STC{cond}{L} Pn,Cd,<Address>  ;Store to memory (from coprocessor)
          0: STC2{L}      Pn,Cd,<Address>  ;Store to memory (from coprocessor)
          1: LDC{cond}{L} Pn,Cd,<Address>  ;Read from memory (to coprocessor)
```

```
        1: LDC2{L}      Pn,Cd,<Address>  ;Read from memory (to coprocessor)
        whereas {L} indicates long transfer (Bit 22: N=1)
  19-16  Rn     - ARM Base Register             (R0-R15)     (R15=PC+8)
  15-12  Cd     - Coprocessor src/dest Register  (C0-C15)
  11-8   Pn     - Coprocessor number             (P0-P15)
  7-0    Offset - Unsigned Immediate, step 4     (0-1020, in steps of 4)
```
LDC/STC supported by ARMv2 and up, LDC2/STC2 by ARMv5 and up.
Execution time: (n-1)S+2N+bI, n=number of words transferred.
For details refer to original ARM docs, irrelevant in GBA because no coprocessor exists.

Coprocessor Data Operations (CDP) (without Memory or ARM Register operand)
```
  Bit    Expl.
  31-28  Condition (or 1111b for CDP2 opcode on ARMv5 and up)
  27-24  Must be 1110b for this instruction
         ARM-Opcode (fixed)
           CDP{cond} Pn,<cpopc>,Cd,Cn,Cm{,<cp>}
           CDP2      Pn,<cpopc>,Cd,Cn,Cm{,<cp>}
  23-20  CP Opc - Coprocessor operation code       (0-15)
  19-16  Cn     - Coprocessor operand Register     (C0-C15)
  15-12  Cd     - Coprocessor destination Register (C0-C15)
  11-8   Pn     - Coprocessor number               (P0-P15)
  7-5    CP     - Coprocessor information          (0-7)
  4      Reserved, must be zero (otherwise MCR/MRC opcode)
  3-0    Cm     - Coprocessor operand Register     (C0-C15)
```
CDP supported by ARMv2 and up, CDP2 by ARMv5 and up.
Execution time: 1S+bI, b=number of cycles in coprocessor busy-wait loop.
Return: No flags affected, no ARM-registers used/modified.
For details refer to original ARM docs, irrelevant in GBA because no coprocessor exists.

Coprocessor Double-Register Transfer (MCRR, MRRC) - ARMv5TE and up only
```
  Bit    Expl.
  31-28  Condition
  27-21  Must be 1100010b for this instruction
  20     L - Opcode (Load/Store)
          0: MCRR{cond} Pn,opcode,Rd,Rn,Cm  ;write Rd,Rn to coproc
          0: MCRR2      Pn,opcode,Rd,Rn,Cm  ;write Rd,Rn to coproc
          1: MRRC{cond} Pn,opcode,Rd,Rn,Cm  ;read Rd,Rn from coproc
          1: MRRC2      Pn,opcode,Rd,Rn,Cm  ;read Rd,Rn from coproc
  19-16  Rn - Second source/dest register      (R0-R14)
  15-12  Rd - First source/dest register       (R0-R14)
  11-8   Pn     - Coprocessor number           (P0-P15)
  7-4    CP Opc - Coprocessor operation code   (0-15)
  3-0    Cm     - Coprocessor operand Register (C0-C15)
```
MCRR/MRRC are supported by ARMv5TE and up (not ARMv5, not ARMv5TExP).
MCRR2/MRRC2 are supported by ARMv6 and up.

THUMB Instruction Summary

The table below lists all THUMB mode instructions with clock cycles, affected CPSR flags, Format/chapter number, and description.
Only register R0..R7 can be used in thumb mode (unless R8-15,SP,PC are explicitly mentioned).

Logical Operations
```
  Instruction        Cycles Flags Format Expl.
  MOV Rd,Imm8bit      1S     NZ--  3   Rd=nn
  MOV Rd,Rs           1S     NZ00  2   Rd=Rs+0
  MOV R0..14,R8..15   1S     ----  5   Rd=Rs
  MOV R8..14,R0..15   1S     ----  5   Rd=Rs
  MOV R15,R0..15      2S+1N  ----  5   PC=Rs
```

```
  MVN Rd,Rs           1S     NZ--  4   Rd=NOT Rs
  AND Rd,Rs           1S     NZ--  4   Rd=Rd AND Rs
  TST Rd,Rs           1S     NZ--  4 Void=Rd AND Rs
  BIC Rd,Rs           1S     NZ--  4   Rd=Rd AND NOT Rs
  ORR Rd,Rs           1S     NZ--  4   Rd=Rd OR Rs
  EOR Rd,Rs           1S     NZ--  4   Rd=Rd XOR Rs
  LSL Rd,Rs,Imm5bit   1S     NZc-  1   Rd=Rs SHL nn
  LSL Rd,Rs           1S+1I  NZc-  4   Rd=Rd SHL (Rs AND 0FFh)
  LSR Rd,Rs,Imm5bit   1S     NZc-  1   Rd=Rs SHR nn
  LSR Rd,Rs           1S+1I  NZc-  4   Rd=Rd SHR (Rs AND 0FFh)
  ASR Rd,Rs,Imm5bit   1S     NZc-  1   Rd=Rs SAR nn
  ASR Rd,Rs           1S+1I  NZc-  4   Rd=Rd SAR (Rs AND 0FFh)
  ROR Rd,Rs           1S+1I  NZc-  4   Rd=Rd ROR (Rs AND 0FFh)
  NOP                 1S     ----  5   R8=R8
```
Carry flag affected only if shift amount is non-zero.

Arithmetic Operations and Multiply
```
  Instruction        Cycles Flags Format Expl.
  ADD Rd,Rs,Imm3bit   1S     NZCV  2   Rd=Rs+nn
  ADD Rd,Imm8bit      1S     NZCV  3   Rd=Rd+nn
  ADD Rd,Rs,Rn        1S     NZCV  2   Rd=Rs+Rn
  ADD R0..14,R8..15   1S     ----  5   Rd=Rd+Rs
  ADD R8..14,R0..15   1S     ----  5   Rd=Rd+Rs
  ADD R15,R0..15      2S+1N  ----  5   PC=Rd+Rs
  ADD Rd,PC,Imm8bit*4 1S     ---- 12   Rd=(($+4) AND NOT 2)+nn
  ADD Rd,SP,Imm8bit*4 1S     ---- 12   Rd=SP+nn
  ADD SP,Imm7bit*4    1S     ---- 13   SP=SP+nn
  ADD SP,-Imm7bit*4   1S     ---- 13   SP=SP-nn
  ADC Rd,Rs           1S     NZCV  4   Rd=Rd+Rs+Cy
  SUB Rd,Rs,Imm3Bit   1S     NZCV  2   Rd=Rs-nn
  SUB Rd,Imm8bit      1S     NZCV  3   Rd=Rd-nn
  SUB Rd,Rs,Rn        1S     NZCV  2   Rd=Rs-Rn
  SBC Rd,Rs           1S     NZCV  4   Rd=Rd-Rs-NOT Cy
  NEG Rd,Rs           1S     NZCV  4   Rd=0-Rs
  CMP Rd,Imm8bit      1S     NZCV  3 Void=Rd-nn
  CMP Rd,Rs           1S     NZCV  4 Void=Rd-Rs
  CMP R0-15,R8-15     1S     NZCV  5 Void=Rd-Rs
  CMP R8-15,R0-15     1S     NZCV  5 Void=Rd-Rs
  CMN Rd,Rs           1S     NZCV  4 Void=Rd+Rs
  MUL Rd,Rs           1S+mI  NZx-  4   Rd=Rd*Rs
```

Jumps and Calls
```
  Instruction        Cycles    Flags Format Expl.
  B disp              2S+1N     ---- 18  PC=$+/-2048
  BL disp             3S+1N     ---- 19  PC=$+/-4M, LR=$+5
  B{cond=true} disp   2S+1N     ---- 16  PC=$+/-0..256
  B{cond=false} disp  1S        ---- 16  N/A
  BX R0..15           2S+1N     ----  5  PC=Rs, ARM/THUMB (Rs bit0)
  SWI Imm8bit         2S+1N     ---- 17  PC=8, ARM SVC mode, LR=$+2
  BKPT Imm8bit        ???       ---- 17  ??? ARM9 Prefetch Abort
  BLX disp            ???       ---- ??? ??? ARM9
  BLX R0..R14         ???       ---- ??? ??? ARM9
  POP {Rlist,}PC   (n+1)S+2N+1I ---- 14
  MOV R15,R0..15      2S+1N     ----  5  PC=Rs
  ADD R15,R0..15      2S+1N     ----  5  PC=Rd+Rs
```
The thumb BL instruction occupies two 16bit opcodes, 32bit in total.

Memory Load/Store
```
  Instruction        Cycles    Flags Format Expl.
  LDR  Rd,[Rb,5bit*4] 1S+1N+1I  ----  9  Rd = WORD[Rb+nn]
  LDR  Rd,[PC,8bit*4] 1S+1N+1I  ----  6  Rd = WORD[PC+nn]
  LDR  Rd,[SP,8bit*4] 1S+1N+1I  ---- 11  Rd = WORD[SP+nn]
  LDR  Rd,[Rb,Ro]     1S+1N+1I  ----  7  Rd = WORD[Rb+Ro]
```

```
 LDRB Rd,[Rb,5bit*1] 1S+1N+1I  ----  9  Rd = BYTE[Rb+nn]
 LDRB Rd,[Rb,Ro]     1S+1N+1I  ----  7  Rd = BYTE[Rb+Ro]
 LDRH Rd,[Rb,5bit*2] 1S+1N+1I  ---- 10  Rd = HALFWORD[Rb+nn]
 LDRH Rd,[Rb,Ro]     1S+1N+1I  ----  8  Rd = HALFWORD[Rb+Ro]
 LDSB Rd,[Rb,Ro]     1S+1N+1I  ----  8  Rd = SIGNED_BYTE[Rb+Ro]
 LDSH Rd,[Rb,Ro]     1S+1N+1I  ----  8  Rd = SIGNED_HALFWORD[Rb+Ro]
 STR  Rd,[Rb,5bit*4] 2N        ----  9  WORD[Rb+nn] = Rd
 STR  Rd,[SP,8bit*4] 2N        ---- 11  WORD[SP+nn] = Rd
 STR  Rd,[Rb,Ro]     2N        ----  7  WORD[Rb+Ro] = Rd
 STRB Rd,[Rb,5bit*1] 2N        ----  9  BYTE[Rb+nn] = Rd
 STRB Rd,[Rb,Ro]     2N        ----  7  BYTE[Rb+Ro] = Rd
 STRH Rd,[Rb,5bit*2] 2N        ---- 10  HALFWORD[Rb+nn] = Rd
 STRH Rd,[Rb,Ro]     2N        ----  8  HALFWORD[Rb+Ro]=Rd
 PUSH {Rlist}{LR}    (n-1)S+2N ---- 14
 POP  {Rlist}{PC}              ---- 14  (ARM9: with mode switch)
 STMIA Rb!,{Rlist}   (n-1)S+2N ---- 15
 LDMIA Rb!,{Rlist}   nS+1N+1I  ---- 15
```

THUMB Binary Opcode Format

This table summarizes the position of opcode/parameter bits for THUMB mode instructions, Format 1-19.

```
Form|_15|_14|_13|_12|_11|_10|_9_|_8_|_7_|_6_|_5_|_4_|_3_|_2_|_1_|_0_|
__1_|_0___0___0_|__Op___|_______Offset______|____Rs_____|____Rd_____|Shifted
__2_|_0___0___0___1___1_|_I,_Op_|___Rn/nn___|____Rs_____|____Rd_____|ADD/SUB
__3_|_0___0___1_|__Op___|____Rd_____|_____________Offset____________|Immedi.
__4_|_0___1___0___0___0___0_|______Op_______|____Rs_____|____Rd_____|AluOp
__5_|_0___1___0___0___0___1_|__Op___|Hd_|Hs_|____Rs_____|____Rd_____|HiReg/BX
__6_|_0___1___0___0___1_|____Rd_____|_____________Word______________|LDR PC
__7_|_0___1___0___1_|__Op___|_0_|___Ro______|____Rb_____|____Rd_____|LDR/STR
__8_|_0___1___0___1_|__Op___|_1_|___Ro______|____Rb_____|____Rd_____|""H/SB/SH
__9_|_0___1___1_|__Op___|_______Offset______|____Rb_____|____Rd_____|""{B}
_10_|_1___0___0___0_|Op_|_______Offset______|____Rb_____|____Rd_____|""H
_11_|_1___0___0___1_|Op_|____Rd_____|_____________Word______________|"" SP
_12_|_1___0___1___0_|Op_|____Rd_____|_____________Word______________|ADD PC/SP
_13_|_1___0___1___1___0___0___0___0_|_S_|___________Word______________|ADD SP,nn
_14_|_1___0___1___1_|Op_|_1___0_|_R_|_____________Rlist_____________|PUSH/POP
_17_|_1___0___1___1___1___1___1___0_|___________User_Data___________|ARM9:BKPT
_X__|_1___0___1___1___   ..............................             |ARM11...
 0110 011 Change Processor State        CPS on page B4-2
 0001 xxx Compare and Branch on Zero    CBNZ, CBZ on page A6-52
 1011 xxx Compare and Branch on Nonzero CBNZ, CBZ on page A6-52
 0011 xxx Compare and Branch on Zero    CBNZ, CBZ on page A6-52
 1001 xxx Compare and Branch on Nonzero CBNZ, CBZ on page A6-52
 0010 00x Signed Extend Halfword        SXTH on page A6-256
 0010 01x Signed Extend Byte            SXTB on page A6-254
 0010 10x Unsigned Extend Halfword      UXTH on page A6-274
 0010 11x Unsigned Extend Byte          UXTB on page A6-272
 1010 00x Byte-Reverse Word             REV on page A6-191
 1010 01x Byte-Reverse Packed Halfword  REV16 on page A6-192
 1010 11x Byte-Reverse Signed Halfword  REVSH on page A6-193
 1111 xxx If-Then, and hints If-Then, and hints on page A5-11
_15_|_1___1___0___0_|Op_|____Rb_____|_____________Rlist_____________|STM/LDM
_16_|_1___1___0___1_|_____Cond______|__________Signed_Offset_________|B{cond}
_U__|_1___1___0___1___1___1___1___0_|_____________var_______________|UndefARM9
_17_|_1___1___0___1___1___1___1___1_|___________User_Data___________|SWI
_18_|_1___1___1___0___0_|_________________Offset_____________________|B
_19_|_1___1___1___1___0___1_|_____________________________var_____|_0_|BLX.ARM9
_U__|_1___1___1___1___0___1_|_____________________________var_____|_1_|UndefARM9
_19_|_1___1___1___1___1_|_H_|_____________Offset_Low/High_____________|BL,BLX
```

Further UNDEFS ??? ARM9?

```
 1011 0001 xxxxxxxx (reserved)
 1011 0x1x xxxxxxxx (reserved)
 1011 10xx xxxxxxxx (reserved)
```

```
1011 1111 xxxxxxxx (reserved)
1101 1110 xxxxxxxx (free for user)
```

THUMB Opcodes: Register Operations (ALU, BX)

THUMB.1: move shifted register

```
  15-13  Must be 000b for 'move shifted register' instructions
  12-11  Opcode
           00b: LSL{S} Rd,Rs,#Offset   (logical/arithmetic shift left)
           01b: LSR{S} Rd,Rs,#Offset   (logical    shift right)
           10b: ASR{S} Rd,Rs,#Offset   (arithmetic shift right)
           11b: Reserved (used for add/subtract instructions)
  10-6   Offset                      (0-31)
  5-3    Rs - Source register       (R0..R7)
  2-0    Rd - Destination register  (R0..R7)
```

Example: LSL Rd,Rs,#nn ; Rd = Rs << nn ; ARM equivalent: MOVS Rd,Rs,LSL #nn
Zero shift amount is having special meaning (same as for ARM shifts), LSL#0 performs no shift (the carry flag remains unchanged), LSR/ASR#0 are interpreted as LSR/ASR#32. Attempts to specify LSR/ASR#0 in source code are automatically redirected as LSL#0, and source LSR/ASR#32 is redirected as opcode LSR/ASR#0.
Execution Time: 1S
Flags: Z=zeroflag, N=sign, C=carry (except LSL#0: C=unchanged), V=unchanged.

THUMB.2: add/subtract

```
  15-11  Must be 00011b for 'add/subtract' instructions
  10-9   Opcode (0-3)
           0: ADD{S} Rd,Rs,Rn   ;add register        Rd=Rs+Rn
           1: SUB{S} Rd,Rs,Rn   ;subtract register   Rd=Rs-Rn
           2: ADD{S} Rd,Rs,#nn  ;add immediate       Rd=Rs+nn
           3: SUB{S} Rd,Rs,#nn  ;subtract immediate  Rd=Rs-nn
         Pseudo/alias opcode with Imm=0:
           2: MOV{ADDS} Rd,Rs      ;move (affects cpsr) Rd=Rs+0
  8-6    For Register Operand:
           Rn - Register Operand (R0..R7)
         For Immediate Operand:
           nn - Immediate Value  (0-7)
  5-3    Rs - Source register       (R0..R7)
  2-0    Rd - Destination register  (R0..R7)
```

Return: Rd contains result, N,Z,C,V affected (including MOV).
Execution Time: 1S

THUMB.3: move/compare/add/subtract immediate

```
  15-13  Must be 001b for this type of instructions
  12-11  Opcode
           00b: MOV{S} Rd,#nn      ;move     Rd   = #nn
           01b: CMP{S} Rd,#nn      ;compare  Void = Rd - #nn
           10b: ADD{S} Rd,#nn      ;add      Rd   = Rd + #nn
           11b: SUB{S} Rd,#nn      ;subtract Rd   = Rd - #nn
  10-8   Rd - Destination Register  (R0..R7)
  7-0    nn - Unsigned Immediate    (0-255)
```

ARM equivalents for MOV/CMP/ADD/SUB are MOVS/CMP/ADDS/SUBS same format.
Execution Time: 1S
Return: Rd contains result (except CMP), N,Z,C,V affected (for MOV only N,Z).

THUMB.4: ALU operations

```
  15-10  Must be 010000b for this type of instructions
  9-6    Opcode (0-Fh)
           0: AND{S} Rd,Rs     ;AND logical       Rd = Rd AND Rs
           1: EOR{S} Rd,Rs     ;XOR logical       Rd = Rd XOR Rs
           2: LSL{S} Rd,Rs     ;log. shift left   Rd = Rd << (Rs AND 0FFh)
```

```
          3: LSR{S} Rd,Rs     ;log. shift right  Rd = Rd >> (Rs AND 0FFh)
          4: ASR{S} Rd,Rs     ;arit shift right  Rd = Rd SAR (Rs AND 0FFh)
          5: ADC{S} Rd,Rs     ;add with carry    Rd = Rd + Rs + Cy
          6: SBC{S} Rd,Rs     ;sub with carry    Rd = Rd - Rs - NOT Cy
          7: ROR{S} Rd,Rs     ;rotate right      Rd = Rd ROR (Rs AND 0FFh)
          8: TST    Rd,Rs     ;test            Void = Rd AND Rs
          9: NEG{S} Rd,Rs     ;negate            Rd = 0 - Rs
          A: CMP    Rd,Rs     ;compare         Void = Rd - Rs
          B: CMN    Rd,Rs     ;neg.compare     Void = Rd + Rs
          C: ORR{S} Rd,Rs     ;OR logical        Rd = Rd OR Rs
          D: MUL{S} Rd,Rs     ;multiply          Rd = Rd * Rs
          E: BIC{S} Rd,Rs     ;bit clear         Rd = Rd AND NOT Rs
          F: MVN{S} Rd,Rs     ;not               Rd = NOT Rs
  5-3    Rs - Source Register       (R0..R7)
  2-0    Rd - Destination Register  (R0..R7)
```

ARM equivalent for NEG would be RSBS.
Return: Rd contains result (except TST,CMP,CMN),
Affected Flags:

```
  N,Z,C,V for  ADC,SBC,NEG,CMP,CMN
  N,Z,C   for  LSL,LSR,ASR,ROR (carry flag unchanged if zero shift amount)
  N,Z,C   for  MUL on ARMv4 and below: carry flag destroyed
  N,Z     for  MUL on ARMv5 and above: carry flag unchanged
  N,Z     for  AND,EOR,TST,ORR,BIC,MVN
```

Execution Time:

```
  1S      for  AND,EOR,ADC,SBC,TST,NEG,CMP,CMN,ORR,BIC,MVN
  1S+1I   for  LSL,LSR,ASR,ROR
  1S+mI   for  MUL on ARMv4 (m=1..4; depending on MSBs of incoming Rd value)
  1S+mI   for  MUL on ARMv5 (m=3; fucking slow, no matter of MSBs of Rd value)
```

THUMB.5: Hi register operations/branch exchange

```
  15-10  Must be 010001b for this type of instructions
  9-8    Opcode (0-3)
           0: ADD Rd,Rs   ;add        Rd = Rd+Rs
           1: CMP Rd,Rs   ;compare  Void = Rd-Rs  ;CPSR affected
           2: MOV Rd,Rs   ;move       Rd = Rs
           2: NOP         ;nop        R8 = R8
           3: BX  Rs      ;jump       PC = Rs     ;may switch THUMB/ARM
           3: BLX Rs      ;call       PC = Rs     ;may switch THUMB/ARM (ARM9)
  7      MSBd - Destination Register most significant bit (or BL/BLX flag)
  6      MSBs - Source Register most significant bit
  5-3    Rs - Source Register        (together with MSBs: R0..R15)
  2-0    Rd - Destination Register   (together with MSBd: R0..R15)
```

Restrictions: For ADD/CMP/MOV, MSBs and/or MSBd must be set, ie. it is not allowed that both are cleared.
When using R15 (PC) as operand, the value will be the address of the instruction plus 4 (ie. $+4). Except for BX R15: CPU switches to ARM state, and PC is auto-aligned as (($+4) AND NOT 2).
For BX, MSBs may be 0 or 1, MSBd must be zero, Rd is not used/zero.
For BLX, MSBs may be 0 or 1, MSBd must be set, Rd is not used/zero.
For BX/BLX, when Bit 0 of the value in Rs is zero:

```
  Processor will be switched into ARM mode!
  If so, Bit 1 of Rs must be cleared (32bit word aligned).
  Thus, BX PC (switch to ARM) may be issued from word-aligned address
  only, the destination is PC+4 (ie. the following halfword is skipped).
```

BLX may not use R15. BLX saves the return address as LR=PC+3 (with thumb bit).
Using BLX R14 is possible (sets PC=Old_LR, and New_LR=retadr).
Assemblers/Disassemblers should use MOV R8,R8 as NOP (in THUMB mode).
Return: Only CMP affects CPSR condition flags!
Execution Time:

```
 1S     for ADD/MOV/CMP
 2S+1N  for ADD/MOV with Rd=R15, and for BX
```

THUMB Opcodes: Memory Load/Store (LDR/STR)

THUMB.6: load PC-relative (for loading immediates from literal pool)

```
  15-11  Must be 01001b for this type of instructions
  N/A    Opcode (fixed)
           LDR Rd,[PC,#nn]      ;load 32bit    Rd = WORD[PC+nn]
  10-8   Rd - Destination Register   (R0..R7)
  7-0    nn - Unsigned offset        (0-1020 in steps of 4)
```

The value of PC will be interpreted as (($+4) AND NOT 2).
Return: No flags affected, data loaded into Rd.
Execution Time: 1S+1N+1I

THUMB.7: load/store with register offset

```
  15-12  Must be 0101b for this type of instructions
  11-10  Opcode (0-3)
          0: STR  Rd,[Rb,Ro]   ;store 32bit data  WORD[Rb+Ro] = Rd
          1: STRB Rd,[Rb,Ro]   ;store  8bit data  BYTE[Rb+Ro] = Rd
          2: LDR  Rd,[Rb,Ro]   ;load  32bit data  Rd = WORD[Rb+Ro]
          3: LDRB Rd,[Rb,Ro]   ;load   8bit data  Rd = BYTE[Rb+Ro]
  9      Must be zero (0) for this type of instructions
  8-6    Ro - Offset Register              (R0..R7)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.8: load/store sign-extended byte/halfword

```
  15-12  Must be 0101b for this type of instructions
  11-10  Opcode (0-3)
          0: STRH Rd,[Rb,Ro]  ;store 16bit data          HALFWORD[Rb+Ro] = Rd
          1: LDSB Rd,[Rb,Ro]  ;load sign-extended 8bit   Rd = BYTE[Rb+Ro]
          2: LDRH Rd,[Rb,Ro]  ;load zero-extended 16bit  Rd = HALFWORD[Rb+Ro]
          3: LDSH Rd,[Rb,Ro]  ;load sign-extended 16bit  Rd = HALFWORD[Rb+Ro]
  9      Must be set (1) for this type of instructions
  8-6    Ro - Offset Register              (R0..R7)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.9: load/store with immediate offset

```
  15-13  Must be 011b for this type of instructions
  12-11  Opcode (0-3)
          0: STR  Rd,[Rb,#nn]  ;store 32bit data   WORD[Rb+nn] = Rd
          1: LDR  Rd,[Rb,#nn]  ;load  32bit data   Rd = WORD[Rb+nn]
          2: STRB Rd,[Rb,#nn]  ;store  8bit data   BYTE[Rb+nn] = Rd
          3: LDRB Rd,[Rb,#nn]  ;load   8bit data   Rd = BYTE[Rb+nn]
  10-6   nn - Unsigned Offset              (0-31 for BYTE, 0-124 for WORD)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.10: load/store halfword

```
  15-12  Must be 1000b for this type of instructions
  11     Opcode (0-1)
          0: STRH Rd,[Rb,#nn]  ;store 16bit data   HALFWORD[Rb+nn] = Rd
          1: LDRH Rd,[Rb,#nn]  ;load  16bit data   Rd = HALFWORD[Rb+nn]
  10-6   nn - Unsigned Offset              (0-62, step 2)
```

```
  5-3    Rb - Base Register               (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```
Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.11: load/store SP-relative
```
  15-12  Must be 1001b for this type of instructions
  11     Opcode (0-1)
          0: STR  Rd,[SP,#nn]  ;store 32bit data   WORD[SP+nn] = Rd
          1: LDR  Rd,[SP,#nn]  ;load  32bit data   Rd = WORD[SP+nn]
  10-8   Rd - Source/Destination Register  (R0..R7)
  7-0    nn - Unsigned Offset              (0-1020, step 4)
```
Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB Opcodes: Memory Addressing (ADD PC/SP)

THUMB.12: get relative address
```
  15-12  Must be 1010b for this type of instructions
  11     Opcode/Source Register (0-1)
          0: ADD  Rd,PC,#nn    ;Rd = (($+4) AND NOT 2) + nn
          1: ADD  Rd,SP,#nn    ;Rd = SP + nn
  10-8   Rd - Destination Register         (R0..R7)
  7-0    nn - Unsigned Offset              (0-1020, step 4)
```
Return: No flags affected, result in Rd.
Execution Time: 1S

THUMB.13: add offset to stack pointer
```
  15-8   Must be 10110000b for this type of instructions
  7      Opcode/Sign
          0: ADD  SP,#nn       ;SP = SP + nn
          1: ADD  SP,#-nn      ;SP = SP - nn
  6-0    nn - Unsigned Offset    (0-508, step 4)
```
Return: No flags affected, SP adjusted.
Execution Time: 1S

THUMB Opcodes: Memory Multiple Load/Store (PUSH/POP and LDM/STM)

THUMB.14: push/pop registers
```
  15-12  Must be 1011b for this type of instructions
  11     Opcode (0-1)
          0: PUSH {Rlist}{LR}   ;store in memory, decrements SP (R13)
          1: POP  {Rlist}{PC}   ;load from memory, increments SP (R13)
  10-9   Must be 10b for this type of instructions
  8      PC/LR Bit (0-1)
          0: No
          1: PUSH LR (R14), or POP PC (R15)
  7-0    Rlist - List of Registers (R7..R0)
```
In THUMB mode stack is always meant to be 'full descending', ie. PUSH is equivalent to 'STMFD/STMDB' and POP to 'LDMFD/LDMIA' in ARM mode.
Examples:
```
 PUSH {R0-R3}     ;push R0,R1,R2,R3
 PUSH {R0,R2,LR}  ;push R0,R2,LR
 POP  {R4,R7}     ;pop R4,R7
 POP  {R2-R4,PC}  ;pop R2,R3,R4,PC
```

Note: When calling to a sub-routine, the return address is stored in LR register, when calling further sub-routines, PUSH {LR} must be used to save higher return address on stack. If so, POP {PC} can be later used to return from the sub-routine.
POP {PC} ignores the least significant bit of the return address (processor remains in thumb state even if bit0 was cleared), when intending to return with optional mode switch, use a POP/BX combination (eg. POP {R3} / BX R3).
ARM9: POP {PC} copies the LSB to thumb bit (switches to ARM if bit0=0).
Return: No flags affected, SP adjusted, registers loaded/stored.
Execution Time: nS+1N+1I (POP), (n+1)S+2N+1I (POP PC), or (n-1)S+2N (PUSH).

THUMB.15: multiple load/store

```
  15-12  Must be 1100b for this type of instructions
  11     Opcode (0-1)
          0: STMIA Rb!,{Rlist}   ;store in memory, increments Rb
          1: LDMIA Rb!,{Rlist}   ;load from memory, increments Rb
  10-8   Rb - Base register (modified) (R0-R7)
  7-0    Rlist - List of Registers     (R7..R0)
```

Both STM and LDM are incrementing the Base Register.
The lowest register in the list (ie. R0, if it's in the list) is stored/loaded at the lowest memory address.
Examples:

```
 STMIA R7!,{R0-R2}  ;store R0,R1,R2
 LDMIA R0!,{R1,R5}  ;store R1,R5
```

Return: No flags affected, Rb adjusted, registers loaded/stored.
Execution Time: nS+1N+1I for LDM, or (n-1)S+2N for STM.

Strange Effects on Invalid Rlist's
Empty Rlist: R15 loaded/stored (ARMv4 only), and Rb=Rb+40h (ARMv4-v5).
Writeback with Rb included in Rlist: Store OLD base if Rb is FIRST entry in Rlist, otherwise store NEW base (STM/ARMv4), always store OLD base (STM/ARMv5), no writeback (LDM/ARMv4/ARMv5; at this point, THUMB opcodes work different than ARM opcodes).

THUMB Opcodes: Jumps and Calls

THUMB.16: conditional branch

```
  15-12  Must be 1101b for this type of instructions
  11-8   Opcode/Condition (0-Fh)
          0: BEQ label        ;Z=1         ;equal (zero) (same)
          1: BNE label        ;Z=0         ;not equal (nonzero) (not same)
          2: BCS/BHS label    ;C=1         ;unsigned higher or same (carry set)
          3: BCC/BLO label    ;C=0         ;unsigned lower (carry cleared)
          4: BMI label        ;N=1         ;signed  negative (minus)
          5: BPL label        ;N=0         ;signed  positive or zero (plus)
          6: BVS label        ;V=1         ;signed  overflow (V set)
          7: BVC label        ;V=0         ;signed  no overflow (V cleared)
          8: BHI label        ;C=1 and Z=0 ;unsigned higher
          9: BLS label        ;C=0 or Z=1  ;unsigned lower or same
          A: BGE label        ;N=V         ;signed greater or equal
          B: BLT label        ;N<>V        ;signed less than
          C: BGT label        ;Z=0 and N=V ;signed greater than
          D: BLE label        ;Z=1 or N<>V ;signed less or equal
          E: Undefined, should not be used
          F: Reserved for SWI instruction (see SWI opcode)
  7-0    Signed Offset, step 2 ($+4-256..$+4+254)
```

Destination address must by halfword aligned (ie. bit 0 cleared)
Return: No flags affected, PC adjusted if condition true
Execution Time:

```
  2S+1N   if condition true (jump executed)
```

```
  1S      if condition false
```

BX and ADD/MOV PC
See also THUMB.5: BX Rs, and ADD/MOV PC,Rs.

THUMB.18: unconditional branch
```
  15-11  Must be 11100b for this type of instructions
  N/A    Opcode (fixed)
          B label   ;branch (jump)
  10-0   Signed Offset, step 2 ($+4-2048..$+4+2046)
```
Return: No flags affected, PC adjusted.
Execution Time: 2S+1N

THUMB.19: long branch with link
This may be used to call (or jump) to a subroutine, return address is saved in LR (R14).
Unlike all other THUMB mode instructions, this instruction occupies 32bit of memory which are split into two 16bit THUMB opcodes.
```
 First Instruction - LR = PC+4+(nn SHL 12)
  15-11  Must be 11110b for BL/BLX type of instructions
  10-0   nn - Upper 11 bits of Target Address
 Second Instruction - PC = LR + (nn SHL 1), and LR = PC+2 OR 1 (and BLX: T=0)
  15-11  Opcode
          11111b: BL label   ;branch long with link
          11101b: BLX label  ;branch long with link switch to ARM mode (ARM9)
  10-0   nn - Lower 11 bits of Target Address (BLX: Bit0 Must be zero)
```
The destination address range is (PC+4)-400000h..+3FFFFEh, ie. PC+/-4M.
Target must be halfword-aligned. As Bit 0 in LR is set, it may be used to return by a BX LR instruction (keeping CPU in THUMB mode).
Return: No flags affected, PC adjusted, return address in LR.
Execution Time: 3S+1N (first opcode 1S, second opcode 2S+1N).
Note: Exceptions may or may not occur between first and second opcode, this is "implementation defined" (unknown how this is implemented in GBA and NDS).
Using only the 2nd half of BL as "BL LR+imm" is possible (for example, Mario Golf Advance Tour for GBA uses opcode F800h as "BL LR+0").

THUMB.17: software interrupt and breakpoint
SWI supposed for calls to the operating system - Enter Supervisor mode (SVC) in ARM state. BKPT intended for debugging - enters Abort mode in ARM state via Prefetch Abort vector.
```
  15-8   Opcode
          11011111b: SWI nn   ;software interrupt
          10111110b: BKPT nn  ;software breakpoint (ARMv5 and up)
  7-0    nn - Comment Field, ignored by processor (8bit value) (0-255)
```
Execution Time: 2S+1N
The exception handler may interprete the SWI Comment Field by examining the lower 8bit of the 16bit opcode opcode at [R14_svc-2].
If your are also using SWI's from inside of ARM mode, then the SWI handler must examine the T Bit SPSR_svc in order to determine whether it's been a ARM SWI - and if so, examine the lower 24bit of the 32bit opcode opcode at [R14_svc-4].
For Returning from SWI use "MOVS PC,R14", that instruction does restore both PC and CPSR, ie. PC=R14_svc, and CPSR=SPSR_svc, and (as called from THUMB mode), it'll also restore THUMB mode.
Nesting SWIs: SPSR_svc and R14_svc should be saved on stack before either invoking nested SWIs, or (if the IRQ handler uses SWIs) before enabling IRQs.
Execution SWI/BKPT:
```
  R14_svc=PC+2     R14_abt=PC+4   ;save return address
  SPSR_svc=CPSR    SPSR_abt=CPSR  ;save CPSR flags
  CPSR=<changed>   CPSR=<changed> ;Enter svc/abt, ARM state, IRQs disabled
  PC=VVVV0008h     PC=VVVV000Ch   ;jump to SWI/PrefetchAbort vector address
```

THUMB Opcodes: New THUMB Opcodes in ARM11

THUMB.X: signed/unsigned extend byte/halfword (ARMv6)
These unneccessary opcodes are apparently meant to reduce typical compiler overload (eg. for ensuring that "for i=0 to 7" won't end up with i>255).

```
  15-8   Must be 1011.0010b for this type of instructions
  7-6    Opcode
          00b  SXTH  Rd,Rm      ;sign-extend halfword
          01b  SXTB  Rd,Rm      ;sign-extend byte
          10b  UXTH  Rd,Rm      ;zero-extend halfword
          11b  UXTB  Rd,Rm      ;zero-extend byte
  5-3    Rm Source Register                  (R0-R7)
  2-0    Rd Destination Register             (R0-R7)
```

Return: No flags affected (though unspecified?).
Execution Time: ?

THUMB.X: byte-reverse word/halfword(s) (ARMv6)

```
  15-8   Must be 1011.1010b for this type of instructions
  7-6    Opcode
          00b  REV   Rd,Rm      ;swap byte3/byte0 and byte2/byte1  ;word
          01b  REV16 Rd,Rm      ;swap byte3/byte2 and byte1/byte0  ;packed half
          11b  REVSH Rd,Rm      ;swap byte1/byte0 and sign-extend  ;signed half
  5-3    Rm Source Register                  (R0-R7)
  2-0    Rd Destination Register             (R0-R7)
```

Return: No flags affected (though unspecified?).
Execution Time: ?

SETEND (ARMv6)

```
  15-4   Must be 101101100101b
  3      Opcode:
           0: SETEND LE   ;clear CPSR.E bit (little endian data access)
           1: SETEND BE   ;set CPSR.E bit (big endian data access)
  2-0    Should be 000b
```

THUMB.X: change processor state (ARMv6)

```
  15-5   Must be 10110110011b
  4      Opcode:
           0: CPSIE {A}{I}{F}   ;Interrupt Enable
           1: CPSID {A}{I}{F}   ;Interrupt Disable
  3      Must be 0b
  2      A  Affect CPSR.A (?)    (0=No, 1=Yes)
  1      I  Affect CPSR.I (IRQ)  (0=No, 1=Yes)
  0      F  Affect CPSR.F (FIQ)  (0=No, 1=Yes)
```

CPY

```
  15-8   Must be 01000110b for CPY Rd,Rm
  7      MSB of Rd
  6      MSB of Rm
  5-3    Rm Source Register
  2-0    Rd Destination Register
```

Move without changing flags. It's called CPY because THUMB syntax didn't distinguish between MOV and MOVS (until UAL syntax was invented, which makes the name CPY irrelevant).
CPY is officially supported in "T variants of ARMv6 and above" (in practice, it should work on ARMv4T and up, but wasn't officially documented until ARMv6).

ARM Pseudo Instructions and Directives

ARM Pseudo Instructions

```
nop             mov r0,r0
ldr Rd,=Imm     ldr Rd,[r15,disp] ;use .pool as parameter field
add Rd,=addr    add/sub Rd,r15,disp
adr Rd,addr     add/sub Rd,r15,disp
adrl Rd,addr    two add/sub opcodes with disp=xx00h+00yyh
mov Rd,Imm      mvn Rd,NOT Imm    ;or vice-versa
and Rd,Rn,Imm   bic Rd,Rn,NOT Imm ;or vice-versa
cmp Rd,Rn,Imm   cmn Rd,Rn,-Imm    ;or vice-versa
add Rd,Rn,Imm   sub Rd,Rn,-Imm    ;or vice-versa
```

All above opcodes may be made conditional by specifying a {cond} field.

THUMB Pseudo Instructions

```
nop             mov r8,r8
ldr Rd,=Imm     ldr Rd,[r15,disp] ;use .pool as parameter field
add Rd,=addr    add Rd,r15,disp
adr Rd,addr     add Rd,r15,disp
mov Rd,Rs       add Rd,Rs,0       ;with Rd,Rs in range r0-r7 each
```

A22i Directives

```
org  adr     assume following code from this address on
.gba         indicate GBA program
.nds         indicate NDS program
.dsi         indicate DSi program
.firm3ds     indicate 3DS program (.firm format)
.fix         fix GBA/NDS/DSi header checksum
.ereader_create_bmp      create GBA e-Reader dotcode .BMP file(s) (bitmaps)
.ereader_create_raw      create GBA e-Reader dotcode .RAW file (useless)
.ereader_create_bin      create GBA e-Reader dotcode .BIN file (smallest)
.ereader_japan_plus      japanese/plus     (default is non-japanese)
.ereader_japan_original  japanese/original (with Z80-stub for GBA-code)
.title 'Txt' defines a title (used for e-Reader dotcodes)
.teak        select TeakLiteII instruction set (for DSi DSP)
.xtensa      select Xtensa instruction set (for DSi Atheros Wifi)
.rl78        select RL78 instruction set (for 3DS MCU)
.norewrite   do not delete existing output file (keep following data in file)
.data?       following defines RAM data structure (assembled to nowhere)
.code        following is normal ROM code/data (assembled to ROM image)
.include     includes specified source code file (no nesting/error handling)
.import      imports specified binary file (optional parameters: ,begin,len)
.radix nn    changes default numeric format (nn=2,8,10,16 = bin/oct/dec/hex)
.errif expr  generates an error message if expression is nonzero
.if expr     assembles following code only if expression is nonzero
.else        invert previous .if condition
.endif       terminate .if/.ifdef/.ifndef
.ifdef sym   assemble following only if symbol is defined
.ifndef sym  assemble following only if symbol is not defined
.align nn    aligns to an address divisible-by-nn, inserts 00's
.msg         defines a no$gba debugmessage string, such like .msg 'Init Okay'
.brk         defines a no$gba source code break opcode
l equ n      l=n
l:   [cmd]   l=$   (global label)
@@l: [cmd]   @@l=$ (local label, all locals are reset at next global label)
end          end of source code
db ...       define 8bit data (bytes)
dw ...       define 16bit data (halfwords)
dd ...       define 32bit data (words)
defs nn      define nn bytes space (zero-filled)
;...         defines a comment (ignored by the assembler)
//           alias for CRLF, eg. allows <db 'Text',0 // dw addr> in one line
```

A22i Alias Directives (for compatibility with other assemblers)

```
align        .align 4          code16    .thumb
align nn     .align nn         .code 16  .thumb
% nn         defs nn           code32    .arm
.space nn    defs nn           .code 32  .arm
..ds nn      defs nn           ltorg     .pool
x=n          x equ n           .ltorg    .pool
.equ x,n     x equ n           ..ltorg   .pool
.define x n  x equ n           dcb       db (8bit data)
incbin       .import           defb      db (8bit data)
@@@...       ;comment          .byte     db (8bit data)
@ ...        ;comment          .ascii    db (8bit string)
@*...        ;comment          dcw       dw (16bit data)
@...         ;comment          defw      dw (16bit data)
.text        .code             .hword    dw (16bit data)
.bss         .data?            dcd       dd (32bit data)
.global      (ignored)         defd      dd (32bit data)
.extern      (ignored)         .long     dd (32bit data)
.thumb_func  (ignored)         .word     dw/dd, don't use
#directive   .directive        .end      end
.fill nn,1,0 defs nn
```

Alias Conditions, Opcodes, Operands

```
hs   cs   ;condition higher or same = carry set
lo   cc   ;condition lower = carry cleared
asl  lsl  ;arithmetic shift left = logical shift left
```

A22i Numeric Formats & Dialects

```
Type          Normal       Alias
Decimal       85           #85  &d85
Hexadecimal   55h          #55h  0x55  #0x55  $55  &h55
Octal         125o         0o125  &o125
Ascii         'U'          "U"
Binary        01010101b    %01010101  0b01010101  &b01010101
Roman         &rLXXXV      (very useful for arrays of kings and chapters)
```

Note: The default numeric format can be changed by the .radix directive (usually 10=decimal). For example, with radix 16, values like "85" and "0101b" are treated as hexadecimal numbers (in that case, decimal and binary numbers can be still defined with prefixes &d and &b).

A22i Numeric Operators Priority

```
Prio  Operator           Aliases
8     (,) brackets
7     +,- sign
6     *,/,MOD,SHL,SHR    MUL,DIV,<<,>>
5     +,- operation
4     EQ,GE,GT,LE,LT,NE  =,>=,>,<=,<,<>,==,!=
3     NOT
2     AND
1     OR,XOR             EOR
```

Operators of same priority are processed from left to right.
Boolean operators (priority 4) return 1=TRUE, 0=FALSE.

A22i Nocash Syntax

Even though A22i does recognize the official ARM syntax, it's also allowing to use friendly code:

```
mov   r0,0ffh         ;no C64-style "#", and no C-style "0x" required
stmia [r7]!,r0,r4-r5  ;square [base] brackets, no fancy {rlist} brackets
mov   r0,cpsr         ;no confusing MSR and MRS (whatever which is which)
mov   r0,p0,0,c0,c0,0 ;no confusing MCR and MRC (whatever which is which)
ldr   r0,[score]      ;allows to use clean brackets for relative addresses
push  rlist           ;alias for stmfd [r13]!,rlist (and same for pop/ldmfd)
label:                ;label definitions recommended to use ":" colons
```

[A22i is the no$gba debug version's built-in source code assembler.]

ARM CP14 ICEbreaker Debug Communications Channel

The ICEbreaker aka EmbeddedICE module may be found in ARM7TDMI and possibly also in other ARM processors. The main functionality of the module relies on external inputs (BREAKPT signal, etc.) being controlled by external debugging hardware. At software side, ICEbreaker contains a Debug Communications Channel (again to access external hardware), which can be accessed as coprocessor 14 via following opcodes:

```
MRC{cond} P14,0,Rd,C0,C0,0  ;Read Debug Comms Control Register
MRC{cond} P14,0,Rd,C1,C0,0  ;Read Debug Comms Data Register
MRC{cond} P14,0,Rd,C2,C0,0  ;Read Debug Comms Status Register
MCR{cond} P14,0,Rd,C1,C0,0  ;Write Debug Comms Data Register
MCR{cond} P14,0,Rd,C2,C0,0  ;Write Debug Comms Status Register
```

The Control register consists of Bit31-28=ICEbreaker version (0001b for ARM7TDMI), Bit27-2=Not specified, Bit0/Bit1=Data Read/Write Status Flags.

The NDS7 and GBA allow to access CP14 (unlike as for CP0..CP13 & CP15, access to CP14 doesn't generate any exceptions), however, the ICEbreaker module appears to be disabled (or completely unimplemented), any reads from P14,0,Rd,C0,C0,0 through P14,7,Rd,C15,C15,7 are simply returning the prefetched opcode value from [$+8]. ICEbreaker might be eventually used and enabled in Nintendo's hardware debuggers, although external breakpoints are reportedly implemented via /FIQ input rather than via ICEbreaker hardware.
The NDS9 doesn't include a CP14 unit (or it is fully disabled), any attempts to access it are causing invalid instruction exceptions.

ARM CP15 System Control Coprocessor

ARM CP15 Overview

CP15
In many ARM CPUs, particulary such with memory control facilities, coprocessor number 15 (CP15) is used as built-in System Control Coprocessor.
CPUs without memory control functions typically don't include a CP15 at all, in that case even an attempt to read the Main ID register will cause an Undefined Instruction exception.

CP15 Opcodes
CP15 can be accessed via MCR and MRC opcodes, with Pn=P15, and <cpopc>=0.

```
MCR{cond} P15,<cpopc>,Rd,Cn,Cm,<cp>   ;move from ARM to CP15
MRC{cond} P15,<cpopc>,Rd,Cn,Cm,<cp>   ;move from CP15 to ARM
```

Rd can be any ARM register in range R0-R14, R15 should not be used with P15.
<cpopc>,Cn,Cm,<cp> are used to select a CP15 register, eg. 0,C0,C0,0 = Main ID Register.
Other coprocessor opcodes (CDP, LDC, STC) cannot be used with P15.

CP15 Register List

```
  Register       Expl.
  0,C0,C0,0      Main ID Register (R)
  0,C0,C0,1      Cache Type and Size (R)
  0,C0,C0,2      TCM Physical Size (R)
  0,C0,C0,3      ARM11: TLB Type Register
  0,C0,C1,0      ARM11: Processor Feature Register 0
  0,C0,C1,1      ARM11: Processor Feature Register 1
  0,C0,C1,2      ARM11: Debug Feature Register 0
  0,C0,C1,3      ARM11: Auxiliary Feature Register 0
  0,C0,C1,4      ARM11: Memory Model Feature Register 0
  0,C0,C1,5      ARM11: Memory Model Feature Register 1
  0,C0,C1,6      ARM11: Memory Model Feature Register 2
  0,C0,C1,7      ARM11: Memory Model Feature Register 3
  0,C0,C2,0      ARM11: Set Attributes Register 0
  0,C0,C2,1      ARM11: Set Attributes Register 1
  0,C0,C2,2      ARM11: Set Attributes Register 2
  0,C0,C2,3      ARM11: Set Attributes Register 3
  0,C0,C2,4      ARM11: Set Attributes Register 4
  0,C0,C2,5      ARM11: Set Attributes Register 5

  0,C1,C0,0      Control Register (R/W, or R=Fixed)
  0,C1,C0,1      ARM11: Auxiliary Control Register
  0,C1,C0,2      ARM11: Coprocessor Access Control Register
  0,C2,C0,0      ARM11: Translation Table Base Register 0
  0,C2,C0,1      ARM11: Translation Table Base Register 1
  0,C2,C0,2      ARM11: Translation Table Base Control Register
  0,C3,C0,0      ARM11: Domain Access Control Register
  0,C5,C0,0      ARM11: Data Fault Status Register
  0,C5,C0,1      ARM11: Instruction Fault Status Register
  0,C6,C0,0      ARM11: Fault Address Register (FAR)
  0,C6,C0,1      ARM11: Watchpoint Fault Address Register (WFAR)
  0,C2,C0,0      PU Cachability Bits for Data/Unified Protection Region
  0,C2,C0,1      PU Cachability Bits for Instruction Protection Region
  0,C3,C0,0      PU Cache Write-Bufferability Bits for Data Protection Regions
  0,C5,C0,0      PU Access Permission Data/Unified Protection Region
  0,C5,C0,1      PU Access Permission Instruction Protection Region
  0,C5,C0,2      PU Extended Access Permission Data/Unified Protection Region
  0,C5,C0,3      PU Extended Access Permission Instruction Protection Region
  0,C6,C0..C7,0  PU Protection Unit Data/Unified Region 0..7
  0,C6,C0..C7,1  PU Protection Unit Instruction Region 0..7
  0,C7,Cm,Op2    Cache Commands and Halt Function (W)
  0,C9,C0,0      Cache Data Lockdown
  0,C9,C0,1      Cache Instruction Lockdown
  0,C9,C1,0      TCM Data TCM Base and Virtual Size
  0,C9,C1,1      TCM Instruction TCM Base and Virtual Size
  0,C13,Cm,Op2   Misc Process ID registers
  0,C15,Cm,Op2   Misc Implementation Defined and Test/Debug registers
```

Data/Unified Registers

Some Cache/PU/TCM registers are declared as "Data/Unified".
That registers are used for Data accesses in case that the CPU contains separate Data and Instruction registers, otherwise the registers are used for both (unified) Data and Instruction accesses.

ARM CP15 ID Codes

C0,C0,0 - Main ID Register (R)

```
  12-15 ARM Era (0=Pre-ARM7, 7=ARM7, other=Post-ARM7)
```

Post-ARM7 Processors

```
  0-3   Revision Number
  4-15  Primary Part Number (Bit12-15 must be other than 0 or 7)
        (eg. 946h for ARM946)
  16-19 Architecture        (1=v4, 2=v4T, 3=v5, 4=v5T, 5=v5TE, 6=v6, ?=v7)
  20-23 Variant Number
  24-31 Implementor         (41h=ARM, 44h=Digital Equipment Corp, 69h=Intel)
```

ARM7 Processors

```
  0-3   Revision Number
  4-15  Primary Part Number (Bit12-15 must be 7)
  16-22 Variant Number
  23    Architecture        (0=v3, 1=v4T)
  24-31 Implementor         (41h=ARM, 44h=Digital Equipment Corp, 69h=Intel)
```

Pre-ARM7 Processors

```
  0-3   Revision Number
  4-11  Processor ID LSBs (30h=ARM3/v2, 60h,61h,62=ARM600,610,620/v3)
  12-31 Processor ID MSBs (fixed, 41560h)
```

Note: On the NDS9, this register is 41059461h (ARMv5TE, ARM946, rev1). NDS7 and GBA don't have CP15s.

C0,C0,1 - Cache Type Register (R)

```
  0-11  Instruction Cache (bits 0-1=len, 2=m, 3-5=assoc, 6-8=size, 9-11=zero)
  12-23 Data Cache        (bits 0-1=len, 2=m, 3-5=assoc, 6-8=size, 9-11=zero)
  24    Separate Cache Flag (0=Unified, 1=Separate Data/Instruction Caches)
  25-28 Cache Type (0,1,2,6,7=see below, other=reserved)
         Type Method         Cache cleaning         Cache lock-down
         0    Write-through  Not needed             Not supported
         1    Write-back     Read data block        Not supported
         2    Write-back     Register 7 operations  Not supported
         6    Write-back     Register 7 operations  Format A
         7    Write-back     Register 7 operations  Format B      ;<-- NDS9
  29-31 Reserved (zero)
```

The 12bit Instruction/Data values are decoded as shown below,

```
  Cache Absent  = (ASSOC=0 and M=1)       ;in that case overriding below
  Cache Size    = 200h+(100h*M) shl SIZE  ;min 0.5Kbytes, max 96Kbytes
  Associativity = (1+(0.5*M)) shl ASSOC   ;min 1-way,     max 192-way
  Line Length   = 8 shl LEN               ;min 8 bytes,   max 64 bytes
```

For Unified cache (Bit 24=0), Instruction and Data values are identical.

Note: On the NDS9, this register is 0F0D2112h (Code=2000h bytes, Data=1000h bytes, assoc=whatver, and line size 32 bytes each). NDS7 and GBA don't have CP15s (nor any code/data cache).

C0,C0,2 - Tightly Coupled Memory (TCM) Size Register (R)

```
  0-1   Reserved    (0)
  2     ITCM Absent (0=Present, 1=Absent)
  3-5   Reserved    (0)
  6-9   ITCM Size   (Size = 512 SHL N) (or 0=None)
  10-13 Reserved    (0)
  14    DTCM Absent (0=Present, 1=Absent)
  15-17 Reserved    (0)
  18-21 DTCM Size   (Size = 512 SHL N) (or 0=None)
  22-31 Reserved    (0)
```

Note: On the NDS9, this register is 00140180h (ITCM=8000h bytes, DTCM=4000h bytes)). NDS7 and GBA don't have CP15s (nor any ITCM/DTCM).

C0,C0,3..7 - Reserved (R)

Unused/Reserved registers, containing the same value as C0,C0,0.

ARM CP15 Control Register

C1,C0,0 - Control Register (R/W, or R=Fixed)

```
0  MMU/PU Enable         (0=Disable, 1=Enable) (Fixed 0 if none)
1  Alignment Fault Check (0=Disable, 1=Enable) (Fixed 0/1 if none/always on)
2  Data/Unified Cache    (0=Disable, 1=Enable) (Fixed 0/1 if none/always on)
3  Write Buffer          (0=Disable, 1=Enable) (Fixed 0/1 if none/always on)
4  Exception Handling    (0=26bit, 1=32bit)    (Fixed 1 if always 32bit)
5  26bit-address faults  (0=Enable, 1=Disable) (Fixed 1 if always 32bit)
6  Abort Model (pre v4)  (0=Early, 1=Late Abort) (Fixed 1 if ARMv4 and up)
7  Endian                (0=Little, 1=Big)     (Fixed 0/1 if fixed)
8  System Protection bit (MMU-only)
9  ROM Protection bit    (MMU-only)
10 Implementation defined
11 Branch Prediction     (0=Disable, 1=Enable)
12 Instruction Cache     (0=Disable, 1=Enable) (ignored if Unified cache)
13 Exception Vectors     (0=00000000h, 1=FFFF0000h)
14 Cache Replacement     (0=Normal/PseudoRandom, 1=Predictable/RoundRobin)
15 Pre-ARMv5 Mode        (0=Normal, 1=Pre ARMv5; LDM/LDR/POP_PC.Bit0/Thumb)
16 DTCM Enable           (0=Disable, 1=Enable)
17 DTCM Load Mode        (0=R/W, 1=DTCM Write-only)
18 ITCM Enable           (0=Disable, 1=Enable)
19 ITCM Load Mode        (0=R/W, 1=ITCM Write-only)
20 Reserved              (0)
21 Reserved              (0)
22 Unaligned Access      (?=Enable unaligned access and mixed endian)
23 Extended Page Table   (0=Subpage AP Bits Enabled, 1=Disabled)
24 Reserved              (0)
25 CPSR E on exceptions  (0=Clear E bit, 1=Set E bit)
26 Reserved              (0)
27 FIQ Behaviour         (0=Normal FIQ behaviour, 1=FIQs behave as NMFI)
28 TEX Remap bit         (0=No remapping, 1=Remap registers used)
29 Force AP              (0=Access Bit not used, 1=AP[0] used as Access bit)
30 Reserved              (0)
31 Reserved              (0)
```

Various bits in this register may be read-only (fixed 0 if unsupported, or fixed 1 if always activated).
On NDS ARM9, bit0,2,7,12..19 are R/W, bit3..6 are always set, all other bits are always zero.
On 3DS ARM11, bit0..2,8..9,11..13,15,22..23,25,28..29 are R/W, bit3..6,14,16,18 are always set, all other bits are always zero.

ARM CP15 Memory Managment Unit (MMU)

Nintendo consoles have a MMU on the ARM11 processor (in 3DS).

The MMU allows to assign virtual memory addresses, and to disable the data cache in sensitive memory areas (eg. I/O area, and memory that is shared for ARM9/ARM11). Additionally, it can trigger page fault exceptions (eg. for error handling or memory mapped files).

Related CP15 registers used in 3DS bootrom MMU init

```
p15,0,c8,c5,0 ;Invalidate Instruction TLB              (=0)
p15,0,c8,c6,0 ;Invalidate Data TLB                     (=0)
p15,0,c2,c0,0 ;Translation Table Base 0 (for process)  (=1FFF4000h+2)
p15,0,c2,c0,1 ;Translation Table Base 1 (for OS and I/O)(=1FFF4000h+2)
p15,0,c2,c0,2 ;Translation Table Base Control          (=0)
p15,0,c3,c0,0 ;Domain Access Control                   (=55555555h)
p15,0,c1,c0,2 ;Coprocessor Access Control (unrelated?) (=0F00000h)
```

MMU Tables
1st Level table (size 4000h) divides 4Gbyte address space into 1Mbyte sections
2nd Level table(s) (size 400h each) divides a 1Mbyte section into 4Kbyte pages

```
_____________________________ ARMv6 Tables _____________________________
```

ARMv6 First-Level Page Table entries

```
   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
   1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0
  |__Ignored_____________________________________|0_0| Fault
  |__Second-Level-Table-Address_________________|P|Domain_|_SBZ_|0_1| Coarse
  |__Address_____________|0|0|n_S_A|_TEX_|AP_|P|Domain_|X_C_B|1_0| Section 1MB
  |__Address______|___SBZ___|1|_SBZ_|_TEX_|AP_|P|Ignored|X_C_B|1_0| Super 16MB
  |_____________________________________________________________|1_1| Reserved
```

BUG: The Supersections do probably also have "n,S,A" flags, probably in bit17-15, too (the 3DS bootrom seems to set those bits as so).

ARMv6 Second-Level Page Table entries

```
   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
   1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0
  |__Ignored_____________________________________________________|0_0| Fault
  |__Address_____________________|X|_TEX_|n_S_A|_SBZ_|AP_|C_B|0_1| Large 64K
  |__Address_____________________________|n_S_A|_TEX_|AP_|C_B|1_X| Small 4K
```

_____ Pre-ARMv6 Tables _____

This is an older format, similar to the newer ARMv6 tables, but without the "n,S,A,X" bits, and, instead, somehow allowing to divide 4K pages into smaller 1K snippets (with separate access privileges in AP0/AP1/AP2/AP3 bits, but without TEX bits).

Backwards-compatible First-Level Page Table entries

```
   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
   1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0
  |__Ignored_____________________________________|0_0| Fault
  |__Second-Level-Table-Address_________________|P|Domain_|_SBZ_|0_1| Coarse
  |__Address_____________|0|0|_SBZ_|_TEX_|AP_|P|Domain_|0_C_B|1_0| Section 1MB
  |__Address______|___SBZ___|1|_SBZ_|_TEX_|AP_|P|Ignored|0_C_B|1_0| Super 16MB
  |_____________________________________________________________|1_1| Reserved
```

Backwards-compatible Second-Level Page Table entries

```
   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
   1_0_9_8_7_6_5_4_3_2_1_0_9_8_7_6_5_4_3_2_1_0
  |__Ignored_____________________________________________________|0_0| Fault
  |__Address_____________________|_?TEX?_|AP3|AP2|AP1|AP0|C_B|0_1| Large 4x16K
  |__Address_____________________________|AP3|AP2|AP1|AP0|C_B|1_0| Small 4x1K
  |__Address_____________________________|_SBZ_|_TEX_|AP_|C_B|1_1| Extended 4K
```

BUG: The 4bit ?TEX? value is probably meant to be a 3bit TEX value, probably with an unused/zero bit; in bit15 or so.

_____ Notes _____

Supersections and Large Pages

16Mbyte Supersections and 64Kbyte Large pages consist of 16 identical table entries. Which is basically same as 16 Sections or 16 Pages with continously increasing entries.

The advantage is internal caching: If one table entry is cached, then the hardware knows that the other 15 nearby entries are supposed to be same; and that they won't need to be loaded into internal cache.

ARM CP15 Protection Unit (PU)

Nintendo consoles have a PU on the ARM9 processor (in NDS, DSi, 3DS).

Protection Unit can be enabled in Bit0 of C1,C0,0 (Control Register).

C2,C0,0 - Cachability Bits for Data/Unified Protection Region (R/W)
C2,C0,1 - Cachability Bits for Instruction Protection Region (if any) (R/W)

```
0-7  Cachable (C) bits for region 0-7
8-31 Reserved/zero
```

C3,C0,0 - Cache Write-Bufferability Bits for Data Protection Regions (R/W)
Allows to select what to do when writing to a cached memory snippet:
Write-Through stores the data in the cache line (so subsequent cache reads return correct data), and additionally writes the data to underlaying memory.
Write-Back stores the data in the cache line only, and marks the line as dirty, but doesn't update the underlaying memory (underlaying memory is updated only when the CPU decides to use the cache line for other purposes, or when the user is manually "Cleaning" the cache line).

```
0-7  Bufferable (B) bits for region 0-7  (0=Write-Through, 1=Write-Back)
8-31 Reserved/zero
```

Instruction fetches are, obviously, always read-operations. So, there are no write-bufferability bits for Instruction Protection Regions.
Note: Unrelated to the "Cache Write-Bufferability", the ARM does also have a "Write Buffer" (a small FIFO that can queue only a few writes).

C5,C0,0 - Access Permission Data/Unified Protection Region (R/W)
C5,C0,1 - Access Permission Instruction Protection Region (if any) (R/W)
C5,C0,2 - Extended Access Permission Data/Unified Protection Region (R/W)
C5,C0,3 - Extended Access Permission Instruction Protection Region (if any) (R/W)
For C5,C0,0 and C5,C0,1:

```
0-15  Access Permission (AP) bits for region 0-7 (Bits 0-1=AP0, 2-3=AP1, etc)
16-31 Reserved/zero
```

For C5,C0,2 and C5,C0,3 (Extended):

```
0-31  Access Permission (AP) bits for region 0-7 (Bits 0-3=AP0, 4-7=AP1, etc)
```

The possible AP settings (0-3 for C5,C0,0..1, or 0-15 for C5,C0,2..3) are:

AP	Privileged	User
0	-	-
1	R/W	-
2	R/W	R
3	R/W	R/W
5	R	-
6	R	R

Settings 5,6 only for Extended Registers, settings 4,7..15 are Reserved.

C6,C0..C7,0 - Protection Unit Data/Unified Region 0..7 (R/W)
C6,C0..C7,1 - Protection Unit Instruction Region 0..7 (R/W) if any

```
0     Protection Region Enable (0=Disable, 1=Enable)
1-5   Protection Region Size   (2 SHL X) ;min=(X=11)=4KB, max=(X=31)=4GB
6-11  Reserved/zero
12-31 Protection Region Base address (Addr = Y*4K; must be SIZE-aligned)
```

Overlapping Regions are allowed, Region 7 is having highest priority, region 0 lowest priority.

Background Region
Additionally, any memory areas outside of the eight Protection Regions are handled as Background Region, this region has neither Read nor Write access.

Unified Region Note
On the NDS, the Region registers are unified (C6,C0..C7,1 are read/write-able mirrors of C6,C0..C7,0). Nethertheless, the Cachabilty and Permission registers are NOT unified (separate registers exists for code and data settings).

ARM CP15 Cache Control

Cache is enabled/controlled by Bit 2,3,12,14 in Control Register.
Cache regions are controlled via Protection Unit (PU).
Cache type can be detected via Cache Type Register.

C7,C0..C15,0..7 - Cache Commands (W)
Write-only Cache Command Register. Cm,Op2 operands used to select a specific command, with parameter value in Rd.

```
  Cn,Cm,Op2 Rd   ARM9 Command
  C7,C0,4   0    Yes  Wait For Interrupt (Halt)
  C7,C5,0   0    Yes  Invalidate Entire Instruction Cache
  C7,C5,1   VA   Yes  Invalidate Instruction Cache Line
  C7,C5,2   S/I  -    Invalidate Instruction Cache Line
  C7,C5,4   0    -    Flush Prefetch Buffer
  C7,C5,6   0    -    Flush Entire Branch Target Cache
  C7,C5,7   IMP? -    Flush Branch Target Cache Entry
  C7,C6,0   0    Yes  Invalidate Entire Data Cache
  C7,C6,1   VA   Yes  Invalidate Data Cache Line
  C7,C6,2   S/I  -    Invalidate Data Cache Line
  C7,C7,0   0    -    Invalidate Entire Unified Cache
  C7,C7,1   VA   -    Invalidate Unified Cache Line
  C7,C7,2   S/I  -    Invalidate Unified Cache Line
  C7,C8,2   0    Yes  Wait For Interrupt (Halt), alternately to C7,C0,4
  C7,C10,1  VA   Yes  Clean Data Cache Line
  C7,C10,2  S/I  Yes  Clean Data Cache Line
  C7,C10,4  0    -    Drain Write Buffer
  C7,C11,1  VA   -    Clean Unified Cache Line
  C7,C11,2  S/I  -    Clean Unified Cache Line
  C7,C13,1  VA   Yes  Prefetch Instruction Cache Line
  C7,C14,1  VA   Yes  Clean and Invalidate Data Cache Line
  C7,C14,2  S/I  Yes  Clean and Invalidate Data Cache Line
  C7,C15,1  VA   -    Clean and Invalidate Unified Cache Line
  C7,C15,2  S/I  -    Clean and Invalidate Unified Cache Line
```

Parameter values (Rd) formats:

```
  0    Not used, should be zero
  VA   Virtual Address
  S/I  Set/index; Bit 31..(32-A) = Index, Bit (L+S-1)..L = Set ?
```

Note:

```
  Invalidate means to forget all data
  Clean means to write-back dirty cache lines to underlaying memory
  (Clean is important when having "Cache Write-Bufferability" enabled in PU)
```

C9,C0,0 - Data Cache Lockdown
C9,C0,1 - Instruction Cache Lockdown
(Width (W) of index field depends on cache ASSOCIATIVETY.)
Format A:

```
  0..(31-W)  Reserved/zero
  (32-W)..31 Lockdown Block Index
```

Format B:

```
  0..(W-1)   Lockdown Block Index
  W..30      Reserved/zero
  31         L
```

Cache/Write-buffer should not be enabled for the whole 4GB memory area, high-speed TCM memory doesn't require caching, and caching would have fatal results on I/O ports. So, cache can be used only in combination with the Protection Unit, which allows to enable/disable caching in specified regions.

Note

ARMv5 instruction set supports a Cache Prepare for Load opcode (PLD), see
ARM Opcodes: Memory: Single Data Transfer (LDR, STR, PLD)

Note
Data cache always requires PU/MMU. Code cache requires PU/MMU on ARM9, but also works without PU/MMU on ARM11.

ARM CP15 Tightly Coupled Memory (TCM)

TCM is high-speed memory, directly contained in the ARM CPU core.

TCM and DMA
TCM doesn't use the ARM bus. A minor disadvantage is that TCM cannot be accessed by DMA. However, the main advantage is that, when using TCM, the CPU can be kept running without any waitstates even while the bus is used for DMA transfers. Operation during DMA works only if all code/data is located in TCM, waitstates are generated if any code/data outside TCM is accessed; in worst case (if there are no gaps in the DMA) then the CPU is halted until the DMA finishes.

TCM and DMA and IRQ
No idea if/how IRQs are handled during DMA? Eventually (unlikely) code in TCM is kept executed until DMA finishes (ie. until the IRQ vector can be accessed. Eventually the IRQ vector is instantly accessed (causing to halt the CPU until DMA finishes). In both cases: Assuming that IRQs are enabled, and that the IRQ vector and/or IRQ handler are located outside TCM.

Separate Instruction (ITCM) and Data (DTCM) Memory
DTCM can be used only for Data accesses, typically used for stacks and other frequently accessed data.
ITCM is primarily intended for instruction accesses, but it can be also used for Data accesses (among others allowing to copy code to ITCM), however, performance isn't optimal when simultaneously accessing ITCM for code and data (such like opcodes in ITCM that use literal pool values in ITCM).

TCM Enable, TCM Load Mode
CP15 Control Register allows to enable ITCM and DTCM, and to switch ITCM/DTCM into Load Mode. In Load Mode (when TCM is enabled), TCM becomes write-only; this allows to read data from source addresses in main memory, and to write data to destination addresses in TCM by using the same addresses; useful for initializing TCM with overlapping source/dest addresses; Load mode works with all Load/Store opcodes, it does NOT work with SWP/SWPB opcodes.

TCM Physical Size can be detected in 3rd ID Code Register. (C0,C0,2)

C9,C1,0 - Data TCM Size/Base (R/W)
C9,C1,1 - Instruction TCM Size/Base (R/W)

```
  0     Reserved     (0)
  1-5   Virtual Size (Size = 512 SHL N) ;min=(N=3)=4KB, max=(N=23)=4GB
  6-11  Reserved     (0)
  12-31 Region Base  (Base = X SHL 12)  ;Base must be Size-aligned
```

The Virtual size settings should be normally same as the Physical sizes (see C0,C0,2). However, smaller sizes are allowed (using only the 1st some KB), as well as bigger sizes (TCM area is then filled with mirrors of physical TCM).
The ITCM region base may be fixed (read-only), for example, on the NDS, ITCM base is always 00000000h, nethertheless the virtual size may be changed (allowing to mirror ITCM to higher addresses).
If DTCM and ITCM do overlap, then ITCM appears to have priority.

TCM and PU

TCM can be used without Protection Unit.
When the protection unit is enabled, TCM is controlled by the PU just like normal memory, the PU should provide R/W Access Permission for TCM regions; cache and write-buffer are not required for high-speed TCM (so both should be disabled for TCM regions).

ARM CP15 Misc

C13,C0,0 - Process ID for Fast Context Switch Extension (FCSE) (R/W)

```
  0-24  Reserved/zero
  25-31 Process ID (PID) (0-127) (0=Disable)
```

The FCSE allows different processes (each assembled with ORG 0) to be located at virtual addresses in the 1st 32MB area. The FCSE splits the total 4GB address space into blocks of 32MB, accesses to Block(0) are redirected to Block(PID):

```
  IF addr<32M then addr=addr+PID*32M
  Respectively, with PID=0, the address remains unchanged (FCSE disabled).
```

The CPU-to-Memory address handling is shown below:

```
  1. CPU outputs a virtual address (VA)
  2. FCSE adjusts the VA to a modified virtual address (MVA)
  3. Cache hits determined by examining the MVA, continue below if no hit
  4. MMU translates MVA to physical address (PA) (if no MMU present: PA=MVA)
  5. Memory access occurs at PA
```

The FCSE allows limited virtual addressing even if no MMU is present.
If the MMU is present, then either the FCSE and/or the MMU can be used for virtual addressing; the advantage of using the FCSE (a single write to C13,C0,0) is less overload; using the MMU for the same purpose would require to change virtual address translation table in memory, and to flush the cache.
The NDS doesn't have a FCSE (the FCSE register is read-only, always zero).

C13,C0,1 - Trace Process ID (R/W)
C13,C1,1 - Trace Process ID (Mirror) (R/W)
This value is output to ETMPROCID pins (if any), allowing to notify external hardware about the currently executed process within multi-tasking programs.

```
  0-31  Process ID
```

C13,C1,1 is a mirror of C13,C0,1 (for compatibility with other ARM processors).
Both registers are read/write-able on NDS9, but there are no external pin-outs.

<cpopc>
Unlike for all other CP15 registers, the <cpopc> operand of the MRC/MCR opcodes isn't always zero for below registers, so below registers are using "cpopc,Cn,Cm,op2" notation (instead of the normal "Cn,Cm,op2" notation).

Built-In-Self-Test (BIST)
Allows to test internal memory (ie. TCM, Cache Memory, and Cache TAGs). The tests are filling (and verifying) the selected memory region thrice (once with the fillvalue, then with the inverted fillvalue, and then again with the fillvalue). The BIST functions are intended for diagnostics purposes only, not for use in normal program code (ARM doesn't guarantee future processors to have backwards compatible BIST functions).

0,C15,C0,1 - BIST TAG Control Register (R/W)
1,C15,C0,1 - BIST TCM Control Register (R/W)
2,C15,C0,1 - BIST Cache Control Register (R/W)

```
  0-15  Data Control (see below)
  16-31 Instruction Control (see below)
```

The above 16bit control values are:

```
  0     Start bit     (Write: 1=Start) (Read: 1=Busy)
  1     Pause bit     (1=Pause)
  2     Enable bit    (1=Enable)
```

```
  3     Fail Flag     (1=Error) (Read Only)
  4     Complete Flag (1=Ready) (Read Only)
  5-15  Size (2^(N+2) bytes) (min=N=1=8bytes, max=N=24=64MB)
```
Size and Pause are not supported in all implementations.
Caution: While and as long as the Enable bit is set, the corresponding memory region(s) will be disabled. Eg. when testing <either> DTCM <and/or> ITCM, <both> DTCM <and> ITCM are forcefully disabled in C1,C0,0 (Control Register), after the test the software must first clear the BIST enable bit, and then restore DTCM/ITCM bits in C1,C0,0. And of course, the content of the tested memory region must be restored when needed.

0,C15,C0,2 - BIST Instruction TAG Address (R/W)
1,C15,C0,2 - BIST Instruction TCM Address (R/W)
2,C15,C0,2 - BIST Instruction Cache Address (R/W)
0,C15,C0,6 - BIST Data TAG Address (R/W)
1,C15,C0,6 - BIST Data TCM Address (R/W)
2,C15,C0,6 - BIST Data Cache Address (R/W)
```
  0-31  Word-aligned Destination Address within Memory Block (eg. within ITCM)
```
On the NDS9, bit0-1, and bit21-31 are always zero.

0,C15,C0,3 - BIST Instruction TAG Fillvalue (R/W)
1,C15,C0,3 - BIST Instruction TCM Fillvalue (R/W)
2,C15,C0,3 - BIST Instruction Cache Fillvalue (R/W)
0,C15,C0,7 - BIST Data TAG Fillvalue (R/W)
1,C15,C0,7 - BIST Data TCM Fillvalue (R/W)
2,C15,C0,7 - BIST Data Cache Fillvalue (R/W)
```
  0-31  Fillvalue for BIST
```
After BIST, the selected memory region is filled by that value. That is, on the NDS9 at least, all words will be filled with the SAME value (ie. NOT with increasing or randomly generated numbers).

0,C15,C0,0 - Cache Debug Test State Register (R/W)
```
  0-8    Reserved (zero)
  9      Disable Instruction Cache Linefill
  10     Disable Data Cache Linefill
  11     Disable Instruction Cache Streaming
  12     Disable Data Cache Streaming
  13-31  Reserved (zero/unpredictable)
```

3,C15,C0,0 - Cache Debug Index Register (R/W)
```
  0..1    Reserved (zero)
  2..4    Word Address
  5..N    Index
  N+1..29 Reserved (zero)
  30..31  Segment
```

3,C15,C1,0 - Cache Debug Instruction TAG (R/W)
3,C15,C2,0 - Cache Debug Data TAG (R/W)
3,C15,C3,0 - Cache Debug Instruction Cache (R/W)
3,C15,C4,0 - Cache Debug Data Cache (R/W)
```
  0..1    Set
  2..3    Dirty Bits
  4       Valid
  5..N    Index
  N+1..31 TAG Address
```

ARM CPU Instruction Cycle Times

Instruction Cycle Summary

```
  Instruction      Cycles      Additional
  ------------------------------------------------------------------
  ALU              1S          +1S+1N if R15 loaded, +1I if SHIFT(Rs)
  MSR,MRS          1S
  LDR              1S+1N+1I    +1S+1N if R15 loaded
  STR              2N
  LDM              nS+1N+1I    +1S+1N if R15 loaded
  STM              (n-1)S+2N
  SWP              1S+2N+1I
  BL (THUMB)       3S+1N
  B,BL             2S+1N
  SWI,trap         2S+1N
  MUL              1S+ml
  MLA              1S+(m+1)I
  MULL             1S+(m+1)I
  MLAL             1S+(m+2)I
  CDP              1S+bI
  LDC,STC          (n-1)S+2N+bI
  MCR              1N+bI+1C
  MRC              1S+(b+1)I+1C
  {cond} false     1S
```

ARM9:

```
  Q{D}ADD/SUB      1S+Interlock.
  CLZ              1S.
  LDR              1S+1N+1L
  LDRB,LDRH,LDRmis 1S+1N+2L
  LDR PC ...
  STR              1S+1N        (not 2N, and both in parallel)
```

Execution Time: 1S+Interlock (SMULxy,SMLAxy,SMULWx,SMLAWx)
Execution Time: 1S+1I+Interlock (SMLALxy)

ARM11:

```
  Observe that Branch Prediction (enabled in CP15 Control register) can affect
  timings for conditional jumps (and presumably non-conditional ones, too).
  Without prediction: A waitloop made of "SUBS+BNE" takes 4 clks per loop cycle
  With prediction: A waitloop made of "SUBS+BNE" takes 2.5 clks per loop cycle
  (apparently the prediction assumes 75% of the jumps to be taken)
```

Whereas,

```
  n = number of words transferred
  b = number of cycles spent in coprocessor busy-wait loop
  m = depends on most significant byte(s) of multiplier operand
```

Above 'trap' is meant to be the execution time for exceptions. And '{cond} false' is meant to be the execution time for conditional instructions which haven't been actually executed because the condition has been false.

The separate meaning of the N,S,I,C cycles is:

N - Non-sequential cycle
Requests a transfer to/from an address which is NOT related to the address used in the previous cycle. (Called 1st Access in GBA language).
The execution time for 1N is 1 clock cycle (plus non-sequential access waitstates).

S - Sequential cycle
Requests a transfer to/from an address which is located directly after the address used in the previous cycle. Ie. for 16bit or 32bit accesses at incrementing addresses, the first access is Non-sequential, the following accesses are sequential. (Called 2nd Access in GBA language).
The execution time for 1S is 1 clock cycle (plus sequential access waitstates).

I - Internal Cycle
CPU is just too busy, not even requesting a memory transfer for now.
The execution time for 1I is 1 clock cycle (without any waitstates).

C - Coprocessor Cycle
The CPU uses the data bus to communicate with the coprocessor (if any), but no memory transfers are requested.

Memory Waitstates
Ideally, memory may be accessed free of waitstates (1N and 1S are then equal to 1 clock cycle each). However, a memory system may generate waitstates for several reasons: The memory may be just too slow. Memory is currently accessed by DMA, eg. sound, video, memory transfers, etc. Or when data is squeezed through a 16bit data bus (in that special case, 32bit access may have more waitstates than 8bit and 16bit accesses). Also, the memory system may separate between S and N cycles (if so, S cycles would be typically faster than N cycles).

Memory Waitstates for Different Memory Areas
Different memory areas (eg. ROM and RAM) may have different waitstates. When executing code in one area which accesses data in another area, then the S+N cycles must be split into code and data accesses: 1N is used for data access, plus (n-1)S for LDM/STM, the remaining S+N are code access. If an instruction jumps to a different memory area, then all code cycles for that opcode are having waitstate characteristics of the NEW memory area (except Thumb BL which still executes 1S in OLD area).

ARM CPU Versions

Version Numbers
ARM CPUs are distributed by name ARM#, and are described as ARMv# in specifications, whereas "#" is NOT the same than "v#", for example, ARM7TDMI is ARMv4TM. That is so confusing, that ARM didn't even attempt to clarify the relationship between the various "#" and "v#" values.

Version Variants
Suffixes like "M" (long multiply), "T" (Thumb support), "E" (Enhanced DSP) indicate presence of special features, additionally to the standard instruction set of a given version, or, when preceded by an "x", indicate the absence of that features.

ARMv1 aka ARM1
Some sort of a beta version, according to ARM never been used in any commercial products.

ARMv2 and up
MUL,MLA
CDP,LDC,MCR,MRC,STC
SWP/SWPB (ARMv2a and up only)
Two new FIQ registers

ARMv3 and up
MRS,MSR opcodes (instead CMP/CMN/TST/TEQ{P} opcodes)
CPSR,SPSR registers (instead PSR bits in R15)
Removed never condition, cond=NV no longer valid
32bit addressing (instead 26bit addressing in older versions)
26bit addressing backwards comptibility mode (except v3G)
Abt and Und modes (instead handling aborts/undefined in Svc mode)
SMLAL,SMULL,UMLAL,UMULL (optionally, INCLUDED in v3M, EXCLUDED in v4xM/v5xM)

ARMv4 aka ARM7 and up
LDRH,LDRSB,LDRSH,STRH
Sys mode (privileged user mode)
BX (only ARMv4T, and any ARMv5 or ARMv5T and up)
THUMB code (only T variants, ie. ARMv4T, ARMv5T)

ARMv5 aka ARM9 and up
BKPT,BLX,CLZ (BKPT,BLX also in THUMB mode)
LDM/LDR/POP PC with mode switch (POP PC also in THUMB mode)
CDP2,LDC2,MCR2,MRC2,STC2 (new coprocessor opcodes)
C-flag unchanged by MUL (instead undefined flag value)
changed instruction cycle timings / interlock ??? or not ???
QADD,QDADD,QDSUB,QSUB opcodes, CPSR.Q flag (v5TE and V5TExP only)
SMLAxy,SMLALxy,SMLAWy,SMULxy,SMULWy (v5TE and V5TExP only)
LDRD,STRD,PLD,MCRR,MRRC (v5TE only, not v5, not v5TExP)
BXJ (jump to Jazelle bytecode) (v5TEJ only)

ARMv6 aka ARM11 and up (see ARM DDI 0100I)
Supports all ARMv5TEJ features (=including Jazelle?).
The following ARM instructions are added:
- CPS, SRS and RFE instructions for improved exception handling
- REV, REV16 and REVSH byte reversal instructions
- SETEND for a revised endian (memory) model
- LDREX and STREX exclusive access instructions
- SXTB, SXTH, UXTB, UXTH byte/halfword extend instructions
- A set of Single Instruction Multiple Data (SIMD) media instructions
- Additional forms of multiply with accumulation into a 64-bit result
The following Thumb instructions are added:
- CPS, CPY (a form of MOV), REV, REV16, REVSH, SETEND, SXTB, SXTH, UXTB, UXTH
SWP/SWPB has been deprecated in ARMv6, made optional in ARMv7 (with the possibility of disabling it if still available), and removed in ARMv8.

ARMv6K aka "ARM11 MPCore" (see ARM DDI 0360F) (used in 3DS)
Unknown how this is abbreviated (ARMv6-M would be something else: Cortex-M).
The MPCore processor provides support for extensions to ARMv6 that include:
- LDREX/STREX{B|H|D} for 8bit/16bit/64bit and CLREX
- HINT (TrueNOP, YIELD, WFI, WFE, SEV)
- Architectural remap registers.
- Revised use of TEX remap bits. The ARMv6 MMU page table descriptors use a large number of bits to describe all of the options for inner and outer cachability. In reality, no application requires all of these options simultaneously. Therefore it is possible to use the TEX remap mechanism to configure the MP11 CPUs to support only a small number of options. This implies a level of indirection in the page table mappings.
The TEX CB encoding table provides two OS managed page table bits. For binary compatibility with existing ARMv6 ports of OSs, this gives a separate mode of operation of the MMU. This is called the TEX Remap configuration and is controlled by bit [28] TR in CP15 Register 1.
- Revised use of AP bits. In the MP11 CPUs the APX and AP[1:0] encoding b111 is Privileged or User mode read-only access. AP[0] indicates an abort type, Access Bit fault, when CP15 c1[29] is 1.
For more information see the ARM Architecture Reference Manual.

ARMv6T2 aka THUMB-2
This doesn't seem to be supported in 3DS. THUMB-2 adds more 2x16bit opcodes in THUMB mode, ie. THUMB code can contain classic 16bit opcodes, mixed with new 2x16bit opcodes. As far as I understand, the old 2x16bit "BL" and "BLX" do no longer exists, or are replaced by differently encode THUMB-2 opcodes?

ARMv6Z
ARMv6K with Security Extensions. Aka TrustZone, see Processor Feature Register 1.

A Milestone in Computer History
Original ARMv2 has been used in the relative rare and expensive Archimedes deluxe home computers in the late eighties, the Archimedes has caught a lot of attention, particularly for being the first home computer that used a BIOS being programmed in BASIC language - which has been a absolutely revolutionary decadency at that time.
Inspired, programmers all over the world have successfully developed even slower and much more inefficient programming languages, which are nowadays consequently used by nearly all ARM programmers, and by most non-ARM programmers as well.

ARM CPU Data Sheet

This present document is an attempt to supply a brief ARM7TDMI reference, hopefully including all information which is relevant for programmers.

Some details that I have treated as meaningless for GBA programming aren't included - such like Big Endian format, and Virtual Memory data aborts, and most of the chapters listed below.

Have a look at the complete data sheet (URL see below) for more detailed verbose information about ARM7TDMI instructions. That document also includes:

```
- Signal Description
  Pins of the original CPU, probably other for GBA.
- Memory Interface
  Optional virtual memory circuits, etc. not for GBA.
- Coprocessor Interface
  As far as I know, none such in GBA.
- Debug Interface
  For external hardware-based debugging.
- ICEBreaker Module
  For external hardware-based debugging also.
- Instruction Cycle Operations
  Detailed: What happens during each cycle of each instruction.
- DC Parameters (Power supply)
- AC Parameters (Signal timings)
```

The official ARM7TDMI data sheet can be downloaded from ARMs webpage,

```
http://www.arm.com/Documentation/UserMans/PDF/ARM7TDMI.html
```

Be prepared for bloated PDF Format, approx 1.3 MB, about 200 pages.

BIOS Functions

The BIOS includes several System Call Functions which can be accessed by SWI instructions. Incoming parameters are usually passed through registers R0,R1,R2,R3. Outgoing registers R0,R1,R3 are typically containing either garbage, or return value(s). All other registers (R2,R4-R14) are kept unchanged.

Caution
When invoking SWIs from inside of ARM state specify SWI NN*10000h, instead of SWI NN as in THUMB state.

Overview
BIOS Function Summary

BIOS Differences between GBA and NDS functions

All Functions Described
BIOS Arithmetic Functions
BIOS Rotation/Scaling Functions
BIOS Decompression Functions
BIOS Memory Copy
BIOS Halt Functions
BIOS Reset Functions
BIOS Misc Functions
BIOS Multi Boot (Single Game Pak)
BIOS Sound Functions
BIOS SHA1 Functions (DSi only)
BIOS RSA Functions (DSi only)

3DS bootroms
BIOS 3DS Exception Vectors

RAM Usage, BIOS Dumps
BIOS RAM Usage
BIOS Dumping

How BIOS Processes SWIs
SWIs can be called from both within THUMB and ARM mode. In ARM mode, only the upper 8bit of the 24bit comment field are interpreted.
Each time when calling a BIOS function 4 words (SPSR, R11, R12, R14) are saved on Supervisor stack (_svc). Once it has saved that data, the SWI handler switches into System mode, so that all further stack operations are using user stack.
In some cases the BIOS may allow interrupts to be executed from inside of the SWI procedure. If so, and if the interrupt handler calls further SWIs, then care should be taken that the Supervisor Stack does not overflow.

BIOS Function Summary

```
GBA  NDS7 NDS9 DSi7 DSi9 Basic Functions
00h  00h  00h  -    -    SoftReset
01h  -    -    -    -    RegisterRamReset
02h  06h  06h  06h  06h  Halt
03h  07h  -    07h  -    Stop/Sleep
04h  04h  04h  04h  04h  IntrWait       ;DSi7/DSi9: both bugged?
05h  05h  05h  05h  05h  VBlankIntrWait ;DSi7/DSi9: both bugged?
06h  09h  09h  09h  09h  Div
07h  -    -    -    -    DivArm
08h  0Dh  0Dh  0Dh  0Dh  Sqrt
09h  -    -    -    -    ArcTan
0Ah  -    -    -    -    ArcTan2
0Bh  0Bh  0Bh  0Bh  0Bh  CpuSet
0Ch  0Ch  0Ch  0Ch  0Ch  CpuFastSet
0Dh  -    -    -    -    GetBiosChecksum
0Eh  -    -    -    -    BgAffineSet
0Fh  -    -    -    -    ObjAffineSet
GBA  NDS7 NDS9 DSi7 DSi9 Decompression Functions
10h  10h  10h  10h  10h  BitUnPack
11h  11h  11h  11h  11h  LZ77UnCompReadNormalWrite8bit   ;"Wram"
12h  -    -    -    -    LZ77UnCompReadNormalWrite16bit  ;"Vram"
-    -    -    01h  01h  LZ77UnCompReadByCallbackWrite8bit
-    12h  12h  02h  02h  LZ77UnCompReadByCallbackWrite16bit
```

```
-    -    -    19h  19h  LZ77UnCompReadByCallbackWrite16bit (same as above)
13h  -    -    -    -    HuffUnCompReadNormal
-    13h  13h  13h  13h  HuffUnCompReadByCallback
14h  14h  14h  14h  14h  RLUnCompReadNormalWrite8bit     ;"Wram"
15h  -    -    -    -    RLUnCompReadNormalWrite16bit    ;"Vram"
-    15h  15h  15h  15h  RLUnCompReadByCallbackWrite16bit
16h  -    16h  -    16h  Diff8bitUnFilterWrite8bit       ;"Wram"
17h  -    -    -    -    Diff8bitUnFilterWrite16bit      ;"Vram"
18h  -    18h  -    18h  Diff16bitUnFilter
GBA  NDS7 NDS9 DSi7 DSi9 Sound (and Multiboot/HardReset/CustomHalt)
19h  08h  -    08h  -    SoundBias
1Ah  -    -    -    -    SoundDriverInit
1Bh  -    -    -    -    SoundDriverMode
1Ch  -    -    -    -    SoundDriverMain
1Dh  -    -    -    -    SoundDriverVSync
1Eh  -    -    -    -    SoundChannelClear
1Fh  -    -    -    -    MidiKey2Freq
20h  -    -    -    -    SoundWhatever0
21h  -    -    -    -    SoundWhatever1
22h  -    -    -    -    SoundWhatever2
23h  -    -    -    -    SoundWhatever3
24h  -    -    -    -    SoundWhatever4
25h  -    -    -    -    MultiBoot
26h  -    -    -    -    HardReset
27h  1Fh  -    1Fh  -    CustomHalt
28h  -    -    -    -    SoundDriverVSyncOff
29h  -    -    -    -    SoundDriverVSyncOn
2Ah  -    -    -    -    SoundGetJumpList
GBA  NDS7 NDS9 DSi7 DSi9 New NDS Functions
-    03h  03h  03h  03h  WaitByLoop
-    0Eh  0Eh  0Eh  0Eh  GetCRC16
-    0Fh  0Fh  -    -    IsDebugger
-    1Ah  -    1Ah  -    GetSineTable
-    1Bh  -    1Bh  -    GetPitchTable (DSi7: bugged)
-    1Ch  -    1Ch  -    GetVolumeTable
-    1Dh  -    1Dh  -    GetBootProcs (DSi7: only 1 proc)
-    -    1Fh  -    1Fh  CustomPost
GBA  NDS7 NDS9 DSi7 DSi9 New DSi Functions (RSA/SHA1)
-    -    -    20h  20h  RSA_Init_crypto_heap
-    -    -    21h  21h  RSA_Decrypt
-    -    -    22h  22h  RSA_Decrypt_Unpad
-    -    -    23h  23h  RSA_Decrypt_Unpad_OpenPGP_SHA1
-    -    -    24h  24h  SHA1_Init
-    -    -    25h  25h  SHA1_Update
-    -    -    26h  26h  SHA1_Finish
-    -    -    27h  27h  SHA1_Init_update_fin
-    -    -    28h  28h  SHA1_Compare_20_bytes
-    -    -    29h  29h  SHA1_Random_maybe
GBA  NDS7 NDS9 DSi7 DSi9 Invalid Functions
2Bh+ 20h+ 20h+ -    -    Crash (SWI xxh..FFh do jump to garbage addresses)
-    xxh  xxh  -    -    Jump to 0   (on any SWI numbers not listed above)
-    -    -    12h  12h  No function (ignored)
-    -    -    2Bh  2Bh  No function (ignored)
-    -    -    40h+ 40h+ Mirror      (SWI 40h..FFh mirror to 00h..3Fh)
-    -    -    xxh  xxh  Hang        (on any SWI numbers not listed above)
```

Invalid NDS functions: NDS7 SWI 01h, 02h, 0Ah, 16h-19h, 1Eh, and NDS9 SWI 01h, 02h, 07h, 08h, 0Ah, 17h, 19h-1Eh will jump to zero (ie. to the NDS7 reset vector, or to NDS9 unused (usually PU-locked ITCM) memory, which will be both redirected to the debug handler, if any).
Invalid DSi functions: DSi9 SWI 00h, 07h-08h, 0Ah, 0Fh, 17h, 1Ah-1Eh, 2Ah, 2Ch-3Fh do hang in endless loop.

BIOS Differences between GBA and NDS functions

Differences between GBA and NDS BIOS functions
- SoftReset uses different addresses
- SWI numbers for Halt, Stop/Sleep, Div, Sqrt have changed
- Halt destroys r0 on NDS9, IntrWait bugged on NDS9
- CpuFastSet allows 4-byte blocks (nice), but...
- CpuFastSet works very SLOW because of a programming bug (uncool)
- Some of the decompression functions are now using callbacks
- SoundBias uses new delay parameter

And, a number of GBA functions have been removed, and some new NDS functions have been added, see: BIOS Function Summary

BIOS Arithmetic Functions

Div
DivArm
Sqrt
ArcTan
ArcTan2

SWI 06h (GBA) or SWI 09h (NDS7/NDS9/DSi7/DSi9) - Div
Signed Division, r0/r1.
```
  r0  signed 32bit Number
  r1  signed 32bit Denom
```
Return:
```
  r0  Number DIV Denom ;signed
  r1  Number MOD Denom ;signed
  r3  ABS (Number DIV Denom) ;unsigned
```
For example, incoming -1234, 10 should return -123, -4, +123.
The function usually gets caught in an endless loop upon division by zero.
Note: The NDS9 and DSi9 additionally support hardware division, by math coprocessor, accessed via I/O Ports, however, the SWI function is a raw software division.

SWI 07h (GBA) - DivArm
Same as above (SWI 06h Div), but incoming parameters are exchanged, r1/r0 (r0=Denom, r1=number). For compatibility with ARM's library. Slightly slower (3 clock cycles) than SWI 06h.

SWI 08h (GBA) or SWI 0Dh (NDS7/NDS9/DSi7/DSi9) - Sqrt
Calculate square root.
```
  r0   unsigned 32bit number
```
Return:
```
  r0   unsigned 16bit number
```
The result is an integer value, so Sqrt(2) would return 1, to avoid this inaccuracy, shift left incoming number by 2*N as much as possible (the result is then shifted left by 1*N). Ie. Sqrt(2 shl 30) would return 1.41421 shl 15.
Note: The NDS9 and DSi9 additionally support hardware square root calculation, by math coprocessor, accessed via I/O Ports, however, the SWI function is a raw software calculation.

SWI 09h (GBA) - ArcTan
Calculates the arc tangent.
```
  r0   Tan, 16bit (1bit sign, 1bit integral part, 14bit decimal part)
```
Return:
```
  r0   "-PI/2<THETA/<PI/2" in a range of C000h-4000h.
```
Note: there is a problem in accuracy with "THETA<-PI/4, PI/4<THETA".

SWI 0Ah (GBA) - ArcTan2
Calculates the arc tangent after correction processing.
Use this in normal situations.

```
  r0   X, 16bit (1bit sign, 1bit integral part, 14bit decimal part)
  r1   Y, 16bit (1bit sign, 1bit integral part, 14bit decimal part)
```

Return:

```
  r0   0000h-FFFFh for 0<=THETA<2PI.
```

BIOS Rotation/Scaling Functions

BgAffineSet
ObjAffineSet

SWI 0Eh (GBA) - BgAffineSet
Used to calculate BG Rotation/Scaling parameters.

```
  r0   Pointer to Source Data Field with entries as follows:
        s32  Original data's center X coordinate (8bit fractional portion)
        s32  Original data's center Y coordinate (8bit fractional portion)
        s16  Display's center X coordinate
        s16  Display's center Y coordinate
        s16  Scaling ratio in X direction (8bit fractional portion)
        s16  Scaling ratio in Y direction (8bit fractional portion)
        u16  Angle of rotation (8bit fractional portion) Effective Range 0-FFFF
  r1   Pointer to Destination Data Field with entries as follows:
        s16  Difference in X coordinate along same line
        s16  Difference in X coordinate along next line
        s16  Difference in Y coordinate along same line
        s16  Difference in Y coordinate along next line
        s32  Start X coordinate
        s32  Start Y coordinate
  r2   Number of Calculations
```

Return: No return value, Data written to destination address.

SWI 0Fh (GBA) - ObjAffineSet
Calculates and sets the OBJ's affine parameters from the scaling ratio and angle of rotation.
The affine parameters are calculated from the parameters set in Srcp.
The four affine parameters are set every Offset bytes, starting from the Destp address.
If the Offset value is 2, the parameters are stored contiguously. If the value is 8, they match the structure of OAM.
When Srcp is arrayed, the calculation can be performed continuously by specifying Num.

```
  r0   Source Address, pointing to data structure as such:
        s16  Scaling ratio in X direction (8bit fractional portion)
        s16  Scaling ratio in Y direction (8bit fractional portion)
        u16  Angle of rotation (8bit fractional portion) Effective Range 0-FFFF
  r1   Destination Address, pointing to data structure as such:
        s16  Difference in X coordinate along same line
        s16  Difference in X coordinate along next line
        s16  Difference in Y coordinate along same line
        s16  Difference in Y coordinate along next line
  r2   Number of calculations
  r3   Offset in bytes for parameter addresses (2=continuous, 8=OAM)
```

Return: No return value, Data written to destination address.

For both Bg- and ObjAffineSet, Rotation angles are specified as 0-FFFFh (covering a range of 360 degrees), however, the GBA BIOS recurses only the upper 8bit; the lower 8bit may contain a fractional portion, but it is ignored by the BIOS.

BIOS Decompression Functions

BitUnPack
Diff8bitUnFilter
HuffUnComp
LZ77UnComp
RLUnComp

Decompression Read/Write Variants

```
ReadNormal:      Fast (src must be memory mapped)
ReadByCallback:  Slow (src can be non-memory, eg. serial Firmware SPI bus)
Write8bitUnits:  Fast (dest must support 8bit writes, eg. not VRAM)
Write16bitUnits: Slow (dest must be halfword-aligned) (for VRAM)
```

BitUnPack - SWI 10h (GBA/NDS7/NDS9/DSi7/DSi9)

Used to increase the color depth of bitmaps or tile data. For example, to convert a 1bit monochrome font into 4bit or 8bit GBA tiles. The Unpack Info is specified separately, allowing to convert the same source data into different formats.

```
r0  Source Address      (no alignment required)
r1  Destination Address (must be 32bit-word aligned)
r2  Pointer to UnPack information:
     16bit  Length of Source Data in bytes     (0-FFFFh)
     8bit   Width of Source Units in bits      (only 1,2,4,8 supported)
     8bit   Width of Destination Units in bits (only 1,2,4,8,16,32 supported)
     32bit  Data Offset (Bit 0-30), and Zero Data Flag (Bit 31)
    The Data Offset is always added to all non-zero source units.
    If the Zero Data Flag was set, it is also added to zero units.
```

Data is written in 32bit units, Destination can be Wram or Vram. The size of unpacked data must be a multiple of 4 bytes. The width of source units (plus the offset) should not exceed the destination width.
Return: No return value, Data written to destination address.

Diff8bitUnFilterWrite8bit (Wram) - SWI 16h (GBA/NDS9/DSi9)
Diff8bitUnFilterWrite16bit (Vram) - SWI 17h (GBA)
Diff16bitUnFilter - SWI 18h (GBA/NDS9/DSi9)

These aren't actually real decompression functions, destination data will have exactly the same size as source data. However, assume a bitmap or wave form to contain a stream of increasing numbers such like 10..19, the filtered/unfiltered data would be:

```
unfiltered:   10  11  12  13  14  15  16  17  18  19
filtered:     10  +1  +1  +1  +1  +1  +1  +1  +1  +1
```

In this case using filtered data (combined with actual compression algorithms) will obviously produce better compression results.
Data units may be either 8bit or 16bit used with Diff8bit or Diff16bit functions respectively.

```
r0  Source address (must be aligned by 4) pointing to data as follows:
     Data Header (32bit)
       Bit 0-3   Data size (must be 1 for Diff8bit, 2 for Diff16bit)
       Bit 4-7   Type (must be 8 for DiffFiltered)
       Bit 8-31  24bit size after decompression
     Data Units (each 8bit or 16bit depending on used SWI function)
       Data0          ;original data
       Data1-Data0    ;difference data
       Data2-Data1    ;...
       Data3-Data2
       ...
r1  Destination address
```

Return: No return value, Data written to destination address.

HuffUnCompReadNormal - SWI 13h (GBA)
HuffUnCompReadByCallback - SWI 13h (NDS/DSi)
The decoder starts in root node, the separate bits in the bitstream specify if the next node is node0 or node1, if that node is a data node, then the data is stored in memory, and the decoder is reset to the root node. The most often used data should be as close to the root node as possible. For example, the 4-byte string "Huff" could be compressed to 6 bits: 10-11-0-0, with root.0 pointing directly to data "f", and root.1 pointing to a child node, whose nodes point to data "H" and data "u".
Data is written in units of 32bits, if the size of the compressed data is not a multiple of 4, please adjust it as much as possible by padding with 0.
Align the source address to a 4Byte boundary.

```
  r0  Source Address, aligned by 4, pointing to:
       Data Header (32bit)
         Bit0-3   Data size in bit units (normally 4 or 8)
         Bit4-7   Compressed type (must be 2 for Huffman)
         Bit8-31  24bit size of decompressed data in bytes
       Tree Size (8bit)
         Bit0-7   Size of Tree Table/2-1 (ie. Offset to Compressed Bitstream)
       Tree Table (list of 8bit nodes, starting with the root node)
        Root Node and Non-Data-Child Nodes are:
         Bit0-5   Offset to next child node,
                  Next child node0 is at (CurrentAddr AND NOT 1)+Offset*2+2
                  Next child node1 is at (CurrentAddr AND NOT 1)+Offset*2+2+1
         Bit6     Node1 End Flag (1=Next child node is data)
         Bit7     Node0 End Flag (1=Next child node is data)
        Data nodes are (when End Flag was set in parent node):
         Bit0-7   Data (upper bits should be zero if Data Size is less than 8)
       Compressed Bitstream (stored in units of 32bits)
         Bit0-31  Node Bits (Bit31=First Bit)  (0=Node0, 1=Node1)
  r1  Destination Address
  r2  Callback temp buffer      ;\for NDS/DSi "ReadByCallback" variants only
  r3  Callback structure        ;/(see Callback notes below)
```

Return: No return value, Data written to destination address.

LZ77UnCompReadNormalWrite8bit (Wram) - SWI 11h (GBA/NDS7/NDS9/DSi7/DSi9)
LZ77UnCompReadNormalWrite16bit (Vram) - SWI 12h (GBA)
LZ77UnCompReadByCallbackWrite8bit - SWI 01h (DSi7/DSi9)
LZ77UnCompReadByCallbackWrite16bit - SWI 12h (NDS), SWI 02h or 19h (DSi)
Expands LZ77-compressed data. The Wram function is faster, and writes in units of 8bits. For the Vram function the destination must be halfword aligned, data is written in units of 16bits.
CAUTION: Writing 16bit units to [dest-1] instead of 8bit units to [dest] means that reading from [dest-1] won't work, ie. the "Vram" function works only with disp=001h..FFFh, but not with disp=000h.
If the size of the compressed data is not a multiple of 4, please adjust it as much as possible by padding with 0.
Align the source address to a 4-Byte boundary.

```
  r0  Source address, pointing to data as such:
       Data header (32bit)
         Bit 0-3   Reserved
         Bit 4-7   Compressed type (must be 1 for LZ77)
         Bit 8-31  Size of decompressed data
       Repeat below. Each Flag Byte followed by eight Blocks.
       Flag data (8bit)
         Bit 0-7   Type Flags for next 8 Blocks, MSB first
       Block Type 0 - Uncompressed - Copy 1 Byte from Source to Dest
         Bit 0-7   One data byte to be copied to dest
       Block Type 1 - Compressed - Copy N+3 Bytes from Dest-Disp-1 to Dest
         Bit 0-3   Disp MSBs
         Bit 4-7   Number of bytes to copy (minus 3)
         Bit 8-15  Disp LSBs
  r1  Destination address
  r2  Callback parameter        ;\for NDS/DSi "ReadByCallback" variants only
  r3  Callback structure        ;/(see Callback notes below)
```

Return: No return value.

RLUnCompReadNormalWrite8bit (Wram) - SWI 14h (GBA/NDS7/NDS9/DSi7/DSi9)
RLUnCompReadNormalWrite16bit (Vram) - SWI 15h (GBA)
RLUnCompReadByCallbackWrite16bit - SWI 15h (NDS7/NDS9/DSi7/DSi9)
Expands run-length compressed data. The Wram function is faster, and writes in units of 8bits. For the Vram function the destination must be halfword aligned, data is written in units of 16bits.
If the size of the compressed data is not a multiple of 4, please adjust it as much as possible by padding with 0.
Align the source address to a 4Byte boundary.

```
  r0  Source Address, pointing to data as such:
       Data header (32bit)
         Bit 0-3   Reserved
         Bit 4-7   Compressed type (must be 3 for run-length)
         Bit 8-31  Size of decompressed data
       Repeat below. Each Flag Byte followed by one or more Data Bytes.
       Flag data (8bit)
         Bit 0-6   Expanded Data Length (uncompressed N-1, compressed N-3)
         Bit 7     Flag (0=uncompressed, 1=compressed)
       Data Byte(s) - N uncompressed bytes, or 1 byte repeated N times
  r1  Destination Address
  r2  Callback parameter        ;\for NDS/DSi "ReadByCallback" variants only
  r3  Callback structure        ;/(see Callback notes below)
```

Return: No return value, Data written to destination address.

NDS/DSi Decompression Callbacks
On NDS and DSi, the "ReadByCallback" variants are reading source data from callback functions (rather than directly from memory). The callback functions may read normal data from memory, or from other devices, such like directly from the gamepak bus, without storing the source data in memory. The downside is that the callback mechanism makes the function very slow, furthermore, NDS7/NDS9 SWI 12h, 13h, 15h are using THUMB code, and variables on stack, alltogether that makes the whole shit very-very-very slow.

```
  r2 = user defined callback parameter (passed on to Open function)
        (or, for Huffman: pointer to temp buffer, max 200h bytes needed)
  r3 = pointer to callback structure
```

Callback structure (five 32bit pointers to callback functions)

```
  Open_and_get_32bit (eg. LDR r0,[r0], get header)
  Close              (optional, 0=none)
  Get_8bit           (eg. LDRB r0,[r0])
  Get_16bit          (not used)
  Get_32bit          (used by Huffman only)
```

All functions may use ARM or THUMB code (indicated by address bit0). The current source address (r0) is passed to all callback functions. Additionally, the initial destination address (r1), and a user defined parameter (r2) are passed to the Open function. For Huffman r2 must point to a temp buffer (max 200h bytes needed, internally used by the SWI function to make a copy of the huffman tree; needed for random-access to the tree, which wouldn't work with the sequentially reading callbacks).
All functions have return values in r0. The Open function normally returns the first word (containing positive length and type), alternatively it may return a negative error code to abort/reject decompression. The Close function, if it is defined, should return zero (or any positive value), or a negative errorcode. The other functions return raw data, without errorcodes. The SWI returns the length of decompressed data, or the signed errorcode from the Open/Close functions.

BIOS Memory Copy

CpuFastSet
CpuSet

SWI 0Ch (GBA/NDS7/NDS9/DSi7/DSi9) - CpuFastSet

Memory copy/fill in units of 32 bytes. Memcopy is implemented as repeated LDMIA/STMIA [Rb]!,r2-r9 instructions. Memfill as single LDR followed by repeated STMIA [Rb]!,r2-r9.
After processing all 32-byte-blocks, the NDS/DSi additonally processes the remaining words as 4-byte blocks.
BUG: The NDS/DSi uses the fast 32-byte-block processing only for the first N bytes (not for the first N words), so only the first quarter of the memory block is FAST, the remaining three quarters are SLOWLY copied word-by-word.
The length is specifed as wordcount, ie. the number of bytes divided by 4.
On the GBA, the length should be a multiple of 8 words (32 bytes) (otherwise the GBA is forcefully rounding-up the length). On NDS/DSi, the length may be any number of words (4 bytes).

```
  r0    Source address        (must be aligned by 4)
  r1    Destination address   (must be aligned by 4)
  r2    Length/Mode
          Bit 0-20  Wordcount (GBA: rounded-up to multiple of 8 words)
          Bit 24    Fixed Source Address (0=Copy, 1=Fill by WORD[r0])
```

Return: No return value, Data written to destination address.

SWI 0Bh (GBA/NDS7/NDS9/DSi7/DSi9) - CpuSet
Memory copy/fill in units of 4 bytes or 2 bytes. Memcopy is implemented as repeated LDMIA/STMIA [Rb]!,r3 or LDRH/STRH r3,[r0,r5] instructions. Memfill as single LDMIA or LDRH followed by repeated STMIA [Rb]!,r3 or STRH r3,[r0,r5].
The length must be a multiple of 4 bytes (32bit mode) or 2 bytes (16bit mode). The (half)wordcount in r2 must be length/4 (32bit mode) or length/2 (16bit mode), ie. length in word/halfword units rather than byte units.

```
  r0    Source address        (must be aligned by 4 for 32bit, by 2 for 16bit)
  r1    Destination address   (must be aligned by 4 for 32bit, by 2 for 16bit)
  r2    Length/Mode
          Bit 0-20  Wordcount (for 32bit), or Halfwordcount (for 16bit)
          Bit 24    Fixed Source Address (0=Copy, 1=Fill by {HALF}WORD[r0])
          Bit 26    Datasize (0=16bit, 1=32bit)
```

Return: No return value, Data written to destination address.

Note: On GBA, NDS7 and DSi7, these two functions will silently reject to do anything if the source start or end addresses are reaching into the BIOS area. The NDS9 and DSi9 don't have such read-proctections.

BIOS Halt Functions

Halt
IntrWait
VBlankIntrWait
Stop/Sleep
CustomHalt

SWI 02h (GBA) or SWI 06h (NDS7/NDS9/DSi7/DSi9) - Halt
Halts the CPU until an interrupt request occurs. The CPU is switched into low-power mode, all other circuits (video, sound, timers, serial, keypad, system clock) are kept operating.
Halt mode is terminated when any enabled interrupts are requested, that is when (IE AND IF) is not zero, the GBA locks up if that condition doesn't get true. However, the state of CPUs IRQ disable bit in CPSR register, and the IME register are don't care, Halt passes through even if either one has disabled interrupts.
On GBA and NDS7/DSi7, Halt is implemented by writing to HALTCNT, Port 4000301h. On NDS9/DSi9, Halt is implemted by writing to System Control Coprocessor (mov p15,0,c7,c0,4,r0 opcode), this opcode hangs if IME=0.
No parameters, no return value.
(GBA/NDS7/DSi7: all registers unchanged, NDS9/DSi9: R0 destroyed)

SWI 04h (GBA/NDS7/NDS9/DSi7/DSi9) - IntrWait ;DSi7/DSi9=bugged?

Continues to wait in Halt state until one (or more) of the specified interrupt(s) do occur. The function forcefully sets IME=1. When using multiple interrupts at the same time, this function is having less overhead than repeatedly calling the Halt function.

```
  r0    0=Return immediately if an old flag was already set (NDS9: bugged!)
        1=Discard old flags, wait until a NEW flag becomes set
  r1    Interrupt flag(s) to wait for (same format as IE/IF registers)
  r2    DSi7 only: Extra flags (same format as DSi7's IE2/IF2 registers)
```

Caution: When using IntrWait or VBlankIntrWait, the user interrupt handler MUST update the BIOS Interrupt Flags value in RAM; when acknowleding processed interrupt(s) by writing a value to the IF register, the same value should be also ORed to the BIOS Interrupt Flags value, at following memory location:

```
  Host     GBA (16bit)  NDS7 (32bit)  NDS9 (32bit)  DSi7-IF2 (32bit)
  Address  [3007FF8h]   [380FFF8h]    [DTCM+3FF8h]  [380FFC0h]
```

NDS9: BUG: No Discard (r0=0) doesn't work. The function always waits for at least one IRQ to occur (no matter which, including IRQs that are not selected in r1), even if the desired flag was already set. NB. the same bug is also found in the GBA/NDS7 functions, but it's compensated by a second bug, ie. the GBA/NDS7 functions are working okay because their "bug doesn't work".
Return: No return value, the selected flag(s) are automatically reset in BIOS Interrupt Flags value in RAM upon return.
DSi9: BUG: The function tries to enter Halt state via Port 4000301h (which would be okay on ARM7, but it's probably ignored on ARM9, which should normally use CP15 to enter Halt state; if Port 4000301h is really ignored, then the function will "successfully" wait for interrupts, but without actually entering any kind of low power mode).
DSi7: BUG: The function tries to wait for IF and IF2 interrupts, but it does accidently ignore the old IF interrupts, and works only with new IF2 ones.

SWI 05h (GBA/NDS7/NDS9/DSi7/DSi9) - VBlankIntrWait ;DSi7/DSi9=bugged?
Continues to wait in Halt status until a new V-Blank interrupt occurs.
The function sets r0=1 and r1=1 (plus r2=0 on DSi7) and does then execute IntrWait (SWI 04h), see IntrWait for details.
No parameters, no return value.

SWI 03h (GBA) - Stop
Switches the GBA into very low power mode (to be used similar as a screen-saver). The CPU, System Clock, Sound, Video, SIO-Shift Clock, DMAs, and Timers are stopped.
Stop state can be terminated by the following interrupts only (as far as enabled in IE register): Joypad, Game Pak, or General-Purpose-SIO.
"The system clock is stopped so the IF flag is not set."
Preparation for Stop:
Disable Video before implementing Stop (otherwise Video just freezes, but still keeps consuming battery power). Possibly required to disable Sound also? Obviously, it'd be also recommended to disable any external hardware (such like Rumble or Infra-Red) as far as possible.
No parameters, no return value.

SWI 07h (NDS7/DSi7) - Sleep
No info, probably similar as GBA SWI 03h (Stop). Sleep is implemented for ARM7 only, not for ARM9. But maybe the ARM7 function does stop <both> ARM7 and ARM9 (?)

SWI 27h (GBA) or SWI 1Fh (NDS7/DSi7) - CustomHalt (Undocumented)
Writes the 8bit parameter value to HALTCNT, below values are equivalent to Halt and Stop/Sleep functions, other values reserved, purpose unknown.

```
  r2  8bit parameter (GBA: 00h=Halt, 80h=Stop) (NDS7/DSi7: 80h=Halt, C0h=Sleep)
```

No return value.

BIOS Reset Functions

SoftReset
RegisterRamReset
HardReset

SWI 00h (GBA/NDS7/NDS9) - SoftReset
Clears 200h bytes of RAM (containing stacks, and BIOS IRQ vector/flags), initializes system, supervisor, and irq stack pointers, sets R0-R12, LR_svc, SPSR_svc, LR_irq, and SPSR_irq to zero, and enters system mode. Note that the NDS9 stack registers are hardcoded (the DTCM base should be set to the default setting of 0800000h). The NDS9 function additionally flushes caches and write buffer, and sets the CP15 control register to 12078h.

Host	sp_svc	sp_irq	sp_sys	zerofilled area	return address
GBA	3007FE0h	3007FA0h	3007F00h	[3007E00h..3007FFFh]	Flag[3007FFAh]
NDS7	380FFDCh	380FFB0h	380FF00h	[380FE00h..380FFFFh]	Addr[27FFE34h]
NDS9	0803FC0h	0803FA0h	0803EC0h	[DTCM+3E00h..3FFFh]	Addr[27FFE24h]

The NDS7/NDS9 return addresses at [27FFE34h/27FFE24h] are usually containing copies of Cartridge Header [034h/024h] entry points, which may select ARM/THUMB state via bit0. The GBA return address 8bit flag is interpreted as 00h=8000000h (ROM), or 01h-FFh=2000000h (RAM), entered in ARM state.
Note: The reset is applied only to the CPU that has executed the SWI (ie. on the NDS, the other CPU will remain unaffected).
Return: Does not return to calling procedure, instead, loads the above return address into R14, and then jumps to that address by a "BX R14" opcode.

SWI 01h (GBA) - RegisterRamReset
Resets the I/O registers and RAM specified in ResetFlags. However, it does not clear the CPU internal RAM area from 3007E00h-3007FFFh.

```
r0  ResetFlags
     Bit   Expl.
     0     Clear 256K on-board WRAM  ;-don't use when returning to WRAM
     1     Clear 32K on-chip WRAM    ;-excluding last 200h bytes
     2     Clear Palette
     3     Clear VRAM
     4     Clear OAM              ;-zerofilled! does NOT disable OBJs!
     5     Reset SIO registers    ;-switches to general purpose mode!
     6     Reset Sound registers
     7     Reset all other registers (except SIO, Sound)
```

Return: No return value.
Bug: LSBs of SIODATA32 are always destroyed, even if Bit5 of R0 was cleared.
The function always switches the screen into forced blank by setting DISPCNT=0080h (regardless of incoming R0, screen becomes white).

SWI 26h (GBA) - HardReset (Undocumented)
This function reboots the GBA (including for getting through the time-consuming nintendo intro, which is making the function particularly useless and annoying).
Parameters: None. Return: Never/Reboot.
Execution Time: About 2 seconds (!)

BIOS Misc Functions

GetBiosChecksum
WaitByLoop
GetCRC16
IsDebugger
GetSineTable
GetPitchTable

GetVolumeTable
CustomPost
GetBootProcs

SWI 0Dh (GBA) - GetBiosChecksum (Undocumented)
Calculates the checksum of the BIOS ROM (by reading in 32bit units, and adding up these values). IRQ and FIQ are disabled during execution.
The checksum is BAAE187Fh (GBA and GBA SP), or BAAE1880h (NDS/3DS in GBA mode, whereas the only difference is that the byte at [3F0Ch] is changed from 00h to 01h, otherwise the BIOS is 1:1 same as GBA BIOS, it does even include multiboot code).
Parameters: None. Return: r0=Checksum.

SWI 03h (NDS7/NDS9/DSi7/DSi9) - WaitByLoop
Performs a "LOP: SUB R0,1 / BGT LOP" wait loop, the loop is executed in BIOS memory, which provides reliable timings (regardless of the memory waitstates & cache state of the calling procedure). Intended only for short delays (eg. flash memory programming cycles).

```
r0  Delay value (should be in range 1..7FFFFFFFh)
```

Execution time varies for ARM7 vs ARM9. On ARM9 it does also depend on whether ROM is cached, and on DSi it does further depended on the ARM9 CPU clock, and on whether using NDS or DSi BIOS ROM (NDS uses faster THUMB code, whilst DSi uses ARM code, which is slow on uncached ARM9 ROM reads). For example, to get a 1 millisecond delay, use following values:

```
CPU  Clock     Cache   BIOS     Value for 1ms
ARM7 33.51MHz  none    NDS/DSi  r0=20BAh    ;=20BAh  ;-ARM7
ARM9 67.03MHz  on      NDS/DSi  r0=20BAh*2  ;=4174h  ;\ARM9 with cache
ARM9 134.06MHz on      DSi      r0=20BAh*4  ;=82E8h  ;/
ARM9 67.03MHz  off     NDS      r0=20BAh/2  ;=105Dh  ;\
ARM9 67.03MHz  off     DSi      r0=20BAh/4  ;=082Eh  ; ARM9 without cache
ARM9 134.06MHz off     DSi      r0=20BAh/3  ;=0AE8h  ;/
```

Return: No return value.

SWI 0Eh (NDS7/NDS9/DSi7/DSi9) - GetCRC16

```
r0  Initial CRC value (16bit, usually FFFFh)
r1  Start Address   (must be aligned by 2)
r2  Length in bytes (must be aligned by 2)
```

CRC16 checksums can be calculated as such:

```
val[0..7] = C0C1h,C181h,C301h,C601h,CC01h,D801h,F001h,A001h
for i=start to end
  crc=crc xor byte[i]
  for j=0 to 7
    crc=crc shr 1:if carry then crc=crc xor (val[j] shl (7-j))
  next j
next i
```

Return:

```
r0  Calculated 16bit CRC Value
```

Additionally, if the length is nonzero, r3 contains the last processed halfword at [addr+len-2]. Unlike most other NDS7/DSi7 SWI functions (which do reject reading from BIOS memory), this allows to dump the NDS7/DSi7 BIOS (except for the memory region that is locked via BIOSPROT Port 4000308h).

SWI 0Fh (NDS7/NDS9) - IsDebugger
Detects if 4MB (normal) or 8MB (debug version) Main RAM installed.
Caution: Fails on ARM9 when cache is enabled (always returns 8MB state).
Return: r0 = result (0=normal console 4MB, 1=debug version 8MB)
Destroys halfword at [27FFFFAh] (NDS7) or [27FFFF8h] (NDS9)!
The SWI 0Fh function doesn't work stable if it gets interrupted by an interrupt which is calling SWI 0Fh, which would destroy the above halfword scratch value (unless the IRQ handler has saved/restored the halfword).

SWI 1Ah (NDS7/DSi7) - GetSineTable

```
  r0  Index (0..3Fh) (must be in that range, otherwise returns garbage)
```
Return: r0 = Desired Entry (0000h..7FF5h) ;SIN(0 .. 88.6 degrees)*8000h

SWI 1Bh (NDS7/DSi7) - GetPitchTable (DSi7: bugged)
```
  r0  Index (0..2FFh) (must be in that range, otherwise returns garbage)
```
BUG: DSi7 accidently reads from SineTable instead of PitchTable, as workaround for obtaining PitchTable values, one can set "r0=(0..2FFh)-46Ah" on DSi.
Return: r0 = Desired Entry (0000h..FF8Ah) (unsigned)

SWI 1Ch (NDS7/DSi7) - GetVolumeTable
```
  r0  Index (0..2D3h) (must be in that range, otherwise returns garbage)
```
Return: r0 = Desired Entry (00h..7Fh) (unsigned)

SWI 1Fh (NDS9/DSi9) - CustomPost
Writes to the POSTFLG register, probably for use by Firmware boot procedure.
```
  r0  32bit value, to be written to POSTFLG, Port 4000300h
```
Return: No return value.

SWI 1Dh (NDS7/DSi7) - GetBootProcs
Returns addresses of Gamecart boot procedure/interrupt handler, probably for use by Firmware boot procedure. Most of the returned NDS7 functions won't work if the POSTFLG register is set.
The return values are somewhat XORed by each other (on DSi7 most of the values are zero; which does rather negate the XORing effect, and, as a special gimmick, one of the zero values is XORed by incoming r2).

BIOS Multi Boot (Single Game Pak)

MultiBoot

SWI 25h (GBA) - MultiBoot
This function uploads & starts program code to slave GBAs, allowing to launch programs on slave units even if no cartridge is inserted into the slaves (this works because all GBA BIOSes contain built-in download procedures in ROM).
However, the SWI 25h BIOS upload function covers only 45% of the required Transmission Protocol, the other 55% must be coded in the master cartridge (see Transmission Protocol below).
```
  r0  Pointer to MultiBootParam structure
  r1  Transfer Mode (undocumented)
       0=256KHz, 32bit, Normal mode    (fast and stable)
       1=115KHz, 16bit, MultiPlay mode (default, slow, up to three slaves)
       2=2MHz,   32bit, Normal mode    (fastest but maybe unstable)
  Note: HLL-programmers that are using the MultiBoot(param_ptr) macro cannot
  specify the transfer mode and will be forcefully using MultiPlay mode.
```
Return:
```
  r0  0=okay, 1=failed
```
See below for more details.

Multiboot Parameter Structure
Size of parameter structure should be 4Ch bytes (the current GBA BIOS uses only first 44h bytes though). The following entries must be set before calling SWI 25h:
```
  Addr Size Name/Expl.
  14h  1    handshake_data (entry used for normal mode only)
  19h  3    client_data[1,2,3]
  1Ch  1    palette_data
  1Eh  1    client_bit (Bit 1-3 set if child 1-3 detected)
  20h  4    boot_srcp  (typically 8000000h+0C0h)
  24h  4    boot_endp  (typically 8000000h+0C0h+length)
```

The transfer length (excluding header data) should be a multiple of 10h, minimum length 100h, max 3FF40h (ca. 256KBytes). Set palette_data as "81h+color*10h+direction*8+speed*2", or as "0f1h+color*2" for fixed palette, whereas color=0..6, speed=0..3, direction=0..1. The other entries (handshake_data, client_data[1-3], and client_bit) must be same as specified in Transmission Protocol (see below hh,cc,y).

Multiboot Transfer Protocol

Below describes the complete transfer protocol, normally only the Initiation part must be programmed in the master cartridge, the main data transfer can be then performed by calling SWI 25h, the slave program is started after SWI 25h completion.

The ending handshake is normally not required, when using it, note that you will need custom code in BOTH master and slave programs.

```
Times  Send   Receive  Expl.
----------------------Required Transfer Initiation in master program
...    6200   FFFF     Slave not in multiplay/normal mode yet
1      6200   0000     Slave entered correct mode now
15     6200   720x     Repeat 15 times, if failed: delay 1/16s and restart
1      610y   720x     Recognition okay, exchange master/slave info
60h    xxxx   NN0x     Transfer C0h bytes header data in units of 16bits
1      6200   000x     Transfer of header data completed
1      620y   720x     Exchange master/slave info again
...    63pp   720x     Wait until all slaves reply 73cc instead 720x
1      63pp   73cc     Send palette_data and receive client_data[1-3]
1      64hh   73uu     Send handshake_data for final transfer completion
----------------------Below is SWI 25h MultiBoot handler in BIOS
DELAY  -      -        Wait 1/16 seconds at master side
1      llll   73rr     Send length information and receive random data[1-3]
LEN    yyyy   nnnn     Transfer main data block in units of 16 or 32 bits
1      0065   nnnn     Transfer of main data block completed, request CRC
...    0065   0074     Wait until all slaves reply 0075 instead 0074
1      0065   0075     All slaves ready for CRC transfer
1      0066   0075     Signalize that transfer of CRC follows
1      zzzz   zzzz     Exchange CRC must be same for master and slaves
----------------------Optional Handshake (NOT part of master/slave BIOS)
...    ....   ....     Exchange whatever custom data
```

Legend for above Protocol

```
y     client_bit, bit(s) 1-3 set if slave(s) 1-3 detected
x     bit 1,2,or 3 set if slave 1,2,or 3
xxxx  header data, transferred in 16bit (!) units (even in 32bit normal mode)
nn    response value for header transfer, decreasing 60h..01h
pp    palette_data
cc    random client_data[1..3] from slave 1-3, FFh if slave not exists
hh    handshake_data, 11h+client_data[1]+client_data[2]+client_data[3]
uu    random data, not used, ignore this value
```

Below automatically calculated by SWI 25h BIOS function (don't care about)

```
llll  download length/4-34h
rr    random data from each slave for encryption, FFh if slave not exists
yyyy  encoded data in 16bit (multiplay) or 32bit (normal mode) units
nnnn  response value, lower 16bit of destadr in GBA memory (00C0h and up)
zzzz  16bit download CRC value, must be same for master and slaves
```

Pseudo Code for SWI 25h Transfer with Checksum and Encryption calculations

```
if normal_mode    then c=C387h:x=C37Bh:k=43202F2Fh
if multiplay_mode then c=FFF8h:x=A517h:k=6465646Fh
m=dword(pp,cc,cc,cc):f=dword(hh,rr,rr,rr)
for ptr=000000C0h to (file_size-4) step 4
  c=c xor data[ptr]:for i=1 to 32:c=c shr 1:if carry then c=c xor x:next
  m=(6F646573h*m)+1
  send_32_or_2x16 (data[ptr] xor (-2000000h-ptr) xor m xor k)
next
c=c xor f:for i=1 to 32:c=c shr 1:if carry then c=c xor x:next
wait_all_units_ready_for_checksum:send_32_or_1x16 (c)
```

Whereas, explained: c=chksum,x=chkxor,f=chkfin,k=keyxor,m=keymul

Multiboot Communication
In Multiplay mode, master sends 16bit data, and receives 16bit data from each slave (or FFFFh if none). In Normal mode, master sends 32bit data (upper 16bit zero, lower 16bit as for multiplay mode), and receives 32bit data (upper 16bit as for multiplay mode, and lower 16bit same as lower 16bit previously sent by master). Because SIODATA32 occupies same addresses as SIOMULTI0-1, the same transfer code can be used for both multiplay and normal mode (in normal mode SIOMULTI2-3 should be forced to FFFFh though). After each transfer, master should wait for Start bit cleared in SIOCNT register, followed by a 36us delay.
Note: The multiboot slave would also recognize data being sent in Joybus mode, however, master GBAs cannot use joybus mode (because GBA hardware cannot act as master in joybus mode).

Multiboot Slave Header
The transferred Header block is written to 2000000-20000BFh in slave RAM, the header must contain valid data (identically as for normal ROM-cartridge headers, including a copy of the Nintendo logo, correct header CRC, etc.), in most cases it'd be recommended just to transfer a copy of the master cartridges header from 8000000h-80000BFh.

Multiboot Slave Program/Data
The transferred main program/data block is written to 20000C0h and up (max 203FFFFh) in slave RAM, note that absolute addresses in the program must be then originated at 2000000h rather than 8000000h. In case that the master cartridge is 256K or less, it could just transfer a copy of the whole cartridge at 80000C0h and up, the master should then copy & execute its own ROM data into RAM as well.

Multiboot Slave Extended Header
For Multiboot slaves, separate Entry Point(s) must be defined at the beginning of the Program/Data block (the Entry Point in the normal header is ignored), also some reserved bytes in this section are overwritten by the Multiboot procedure. For more information see chapter about Cartridge Header.

Multiboot Slave with Cartridge
Beside for slaves without cartridge, multiboot can be also used for slaves which do have a cartridge inserted, if so, SELECT and START must be kept held down during power-on in order to switch the slave GBA into Multiboot mode (ie. to prevent it from starting the cartridge as normally).
The general idea is to enable newer programs to link to any existing older GBA programs, even if these older programs originally didn't have been intended to support linking.
The uploaded program may access the slaves SRAM, Flash ROM, or EEPROM (if any, allowing to read out or modify slave game positions), as well as cartridge ROM at 80000A0h-8000FFFh (the first 4KBytes, excluding the nintendo logo, allowing to read out the cartridge name from the header, for example).
The main part of the cartridge ROM is meant to be locked out in order to prevent software pirates from uploading "intruder" programs which would send back a copy of the whole cartridge to the master, however, for good or evil, at present time, current GBA models and GBA carts do not seem to contain any such protection.

Uploading Programs from PC
Beside for the ability to upload a program from one GBA to another, this feature can be also used to upload small programs from a PC to a GBA. For more information see chapter about External Connectors.

Nintendo DS
The GBA multiboot function requires a link port, and so, works on GBA and GBA SP only. The Nintendo DS in GBA mode does include the multiboot BIOS function, but it won't be of any use as the DS doesn't have a link port.

BIOS Sound Functions

MidiKey2Freq

SoundBias
SoundChannelClear
SoundDriverInit
SoundDriverMain
SoundDriverMode
SoundDriverVSync
SoundDriverVSyncOff
SoundDriverVSyncOn
SoundWhatever0..4
SoundGetJumpList

SWI 1Fh (GBA) - MidiKey2Freq
Calculates the value of the assignment to ((SoundArea)sa).vchn[x].fr when playing the wave data, wa, with the interval (MIDI KEY) mk and the fine adjustment value (halftones=256) fp.

```
r0  WaveData* wa
r1  u8 mk
r2  u8 fp
```

Return:

```
r0  u32
```

This function is particularly popular because it allows to read from BIOS memory without copy protection range checks. The formula to read one byte (a) from address (i, 0..3FFF) is:
a = (MidiKey2Freq(i-(((i AND 3)+1)OR 3), 168, 0) * 2) SHR 24

SWI 19h (GBA) or SWI 08h (NDS7/DSi7) - SoundBias
Increments or decrements the current level of the SOUNDBIAS register (with short delays) until reaching the desired new level. The upper bits of the register are kept unchanged.

```
r0   BIAS level (0=Level 000h, any other value=Level 200h)
r1   Delay Count (NDS/DSi only) (GBA uses a fixed delay count of 8)
```

Return: No return value.

SWI 1Eh (GBA) - SoundChannelClear
Clears all direct sound channels and stops the sound.
This function may not operate properly when the library which expands the sound driver feature is combined afterwards. In this case, do not use it.
No parameters, no return value.

SWI 1Ah (GBA) - SoundDriverInit
Initializes the sound driver. Call this only once when the game starts up.
It is essential that the work area already be secured at the time this function is called.
You cannot execute this driver multiple times, even if separate work areas have been prepared.

```
r0  Pointer to work area for sound driver, SoundArea structure as follows:
     SoundArea (sa) Structure
      u32    ident      Flag the system checks to see whether the
                        work area has been initialized and whether it
                        is currently being accessed.
      vu8    DmaCount   User access prohibited
      u8     reverb     Variable for applying reverb effects to direct sound
      u16    d1         User access prohibited
      void   (*func)()  User access prohibited
      int    intp       User access prohibited
      void*  NoUse      User access prohibited
      SndCh  vchn[MAX]  The structure array for controlling the direct
                        sound channels (currently 8 channels are
                        available). The term "channel" here does
                        not refer to hardware channels, but rather to
                        virtual constructs inside the sound driver.
      s8     pcmbuf[PCM_BF*2]
     SoundChannel Structure
```

```
  u8          sf      The flag indicating the status of this channel.
                      When 0 sound is stopped.
                      To start sound, set other parameters and
                      then write 80h to here.
                      To stop sound, logical OR 40h for a
                      release-attached off (key-off), or write zero
                      for a pause. The use of other bits is
                      prohibited.
  u8          r1      User access prohibited
  u8          rv      Sound volume output to right side
  u8          lv      Sound volume output to left side
  u8          at      The attack value of the envelope. When the
                      sound starts, the volume begins at zero and
                      increases every 1/60 second. When it
                      reaches 255, the process moves on to the
                      next decay value.
  u8          de      The decay value of the envelope. It is
                      multiplied by "this value/256" every 1/60
                      sec. and when sustain value is reached, the
                      process moves to the sustain condition.
  u8          su      The sustain value of the envelope. The
                      sound is sustained by this amount.
                      (Actually, multiplied by rv/256, lv/256 and
                      output left and right.)
  u8          re      The release value of the envelope. Key-off
                      (logical OR 40h in sf) to enter this state.
                      The value is multiplied by "this value/256"
                      every 1/60 sec. and when it reaches zero,
                      this channel is completely stopped.
  u8          r2[4]   User access prohibited
  u32         fr      The frequency of the produced sound.
                      Write the value obtained with the
                      MidiKey2Freq function here.
  WaveData*   wp      Pointer to the sound's waveform data. The waveform
                      data can be generated automatically from the AIFF
                      file using the tool (aif2agb.exe), so users normally
                      do not need to create this themselves.
  u32         r3[6]   User access prohibited
  u8          r4[4]   User access prohibited
 WaveData Structure
  u16   type    Indicates the data type. This is currently not used.
  u16   stat    At the present time, non-looped (1 shot) waveform
                is 0000h and forward loop is 4000h.
  u32   freq    This value is used to calculate the frequency.
                It is obtained using the following formula:
                sampling rate x 2^((180-original MIDI key)/12)
  u32   loop    Loop pointer (start of loop)
  u32   size    Number of samples (end position)
  s8    data[]  The actual waveform data. Takes (number of samples+1)
                bytes of 8bit signed linear uncompressed data. The last
                byte is zero for a non-looped waveform, and the same
                value as the loop pointer data for a looped waveform.
```

Return: No return value.

SWI 1Ch (GBA) - SoundDriverMain

Main of the sound driver.

Call every 1/60 of a second. The flow of the process is to call SoundDriverVSync, which is explained later, immediately after the V-Blank interrupt.

After that, this routine is called after BG and OBJ processing is executed.

No parameters, no return value.

SWI 1Bh (GBA) - SoundDriverMode

Sets the sound driver operation mode.

```
  r0  Sound driver operation mode
       Bit    Expl.
       0-6    Direct Sound Reverb value (0-127, default=0) (ignored if Bit7=0)
       7      Direct Sound Reverb set (0=ignore, 1=apply reverb value)
       8-11   Direct Sound Simultaneously-produced (1-12 channels, default 8)
       12-15  Direct Sound Master volume (1-15, default 15)
       16-19  Direct Sound Playback Frequency (1-12 = 5734,7884,10512,13379,
              15768,18157,21024,26758,31536,36314,40137,42048, def 4=13379 Hz)
       20-23  Final number of D/A converter bits (8-11 = 9-6bits, def. 9=8bits)
       24-31  Not used.
```
Return: No return value.

SWI 1Dh (GBA) - SoundDriverVSync
An extremely short system call that resets the sound DMA. The timing is extremely critical, so call this function immediately after the V-Blank interrupt every 1/60 second.
No parameters, no return value.

SWI 28h (GBA) - SoundDriverVSyncOff
Due to problems with the main program if the V-Blank interrupts are stopped, and SoundDriverVSync cannot be called every 1/60 a second, this function must be used to stop sound DMA.
Otherwise, even if you exceed the limit of the buffer the DMA will not stop and noise will result.
No parameters, no return value.

SWI 29h (GBA) - SoundDriverVSyncOn
This function restarts the sound DMA stopped with the previously described SoundDriverVSyncOff.
After calling this function, have a V-Blank occur within 2/60 of a second and call SoundDriverVSync.
No parameters, no return value.

SWI 20h..24h (GBA) - SoundWhatever0..4 (Undocumented)
Whatever undocumented sound-related BIOS functions.

SWI 2Ah (GBA) - SoundGetJumpList (Undocumented)
Receives pointers to 36 additional sound-related BIOS functions.
```
  r0  Destination address (must be aligned by 4) (120h bytes buffer)
```

BIOS SHA1 Functions (DSi only)

SHA1_Init(struct)
SHA1_Update(struct,src,srclen)
SHA1_Finish(dst,struct)
SHA1_Init_Update_Finish(dst,src,srclen)
SHA1_Init_Update_Finish_Mess(dst,dstlen,src,srclen)
SHA1_Compare_20_Bytes(src1,src2)
SHA1_Default_Callback(struct,src,len)

SWI 24h (DSi9/DSi7) - SHA1_Init(struct)
Initializes a 64h-byte structure for SHA1 calculations:
```
  [struct+00h] = 67452301h      ;\
  [struct+04h] = EFCDAB89h      ;
  [struct+08h] = 98BADCFEh      ; initial SHA1 checksum value
  [struct+0Ch] = 10325476h      ;
  [struct+10h] = C3D2E1F0h      ;/
  [struct+14h] = 00000000h ;lsw ;\total len in bits, initially zero
  [struct+18h] = 00000000h ;msw ;/
  [struct+1Ch] = uninitialzed   ;-buffer for incomplete fragment (40h bytes)
```

```
  [struct+5Ch] = 00000000h      ;-incomplete fragment size
  if [struct+60h] = 00000000h then [struct+60h] = SHA1_Default_Callback
```
Observe that the incoming [struct+60h] value should be 00000000h, otherwise the default callback isn't installed (using a different callback doesn't make too much sense, and it's probably not done by any DSi programs) (the callback feature might be intended to mount hardware accelleration, or to hook, customize, encrypt, or replace the SHA1 functionality).

SWI 25h (DSi9/DSi7) - SHA1_Update(struct,src,srclen)

This function should be placed between Init and Finish. The Update function can be called multiple times if the source data is split into separate blocks. There's no alignment requirement (though the function works faster if src is 4-byte aligned).

```
  [struct+14h]=[struct+14h]+len*8  ;64bit value ;-raise total len in bits
  if [struct+5Ch]<>0 and [struct+5Ch]+len>=40h  ;\
    for i=[struct+5Ch] to 3Fh                   ; merge old incomplete chunk
      [struct+1Ch+i]=[src], src=src+1, len=len-1; with new data and process it
    SHA1_Callback(struct,struct+1Ch,40h)        ; (if it gives a full chunk)
    [struct+5Ch]=0                              ;/
  if len>=40h then                              ;\process full 40h-byte chunks
    SHA1_Callback(struct,src,len AND NOT 3Fh)   ; (if src isn't 4-byte aligned
    src=src+(len AND NOT 3Fh)                   ; then the DSi BIOS internally
    len=len AND 3Fh                             ;/copies all chunks to struct)
  if len>0 then                                 ;\
    for i=[struct+5Ch] to [struct+5Ch]+len-1    ; memorize remaining bytes
      [struct+1Ch+i]=[src], src=src+1, len=len-1; as incomplete chunk
      [struct+5Ch]=[struct+5Ch]+1               ;/
```

SWI 26h (DSi9/DSi7) - SHA1_Finish(dst,struct)

```
  [total_len]=bswap8byte([struct+14h]) ;get total len in bits in big-endian
  SHA1_Update(struct,value_80h,1)                          ;append end byte
  while [struct+5Ch]<>38h do SHA1_Update(struct,value_00h,1) ;append padding
  SHA1_Update(struct,total_len,8)                          ;append 64bit len
  [struct+14h]=bswap8byte([total_len]) ;restore total len, exclude above update
  [dst+00h]=bswap([struct+00h]  ;msw   ;\
  [dst+04h]=bswap([struct+04h]         ; store SHA1 result at dst
  [dst+08h]=bswap([struct+08h]         ; (in big-endian)
  [dst+0Ch]=bswap([struct+0Ch]         ;
  [dst+10h]=bswap([struct+10h]  ;lsw   ;/
```

SHA1_Default_Callback(struct,src,len)

```
  for j=1 to len/40h
    a=[struct+0], b=[struct+4], c=[struct+8], d=[struct+0Ch], e=[struct+10h]
    for i=0 to 79
      if i=0..15  then w[i] = bswap([src]), src=src+4
      if i=16..79 then w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) rol 1
      if i=0..19  then f=5A827999h + e + (d xor (b and (c xor d)))
      if i=20..39 then f=6ED9EBA1h + e + (b xor c xor d)
      if i=40..59 then f=8F1BBCDCh + e + ((b and c) or (d and (b or c)))
      if i=60..79 then f=CA62C1D6h + e + (b xor c xor d)
      e=d, d=c, c=(b ror 2), b=a, a=f + (a rol 5) + w[i]
    [struct+0]=[struct+0]+a, [struct+4]=[struct+4]+b, [struct+8]=[struct+8]+c
    [struct+0Ch]=[struct+0Ch]+d, [struct+10h]=[struct+10h]+e
```

SWI 27h (DSi9/DSi7) - SHA1_Init_Update_Finish(dst,src,srclen)

```
  [struct+60h]=00000000h  ;want Init to install the default SHA1 callback
  SHA1_Init(struct)
  SHA1_Update(struct,src,srclen)
  SHA1_Finish(dst,struct)
```
Always returns r0=1.

SWI 29h (DSi9/DSi7) - SHA1_Init_Update_Finish_Mess(dst,dstlen,src,srclen)

```
  if dst=0 then exit(r0=1)  ;uh, that's same return value as when okay
```

```
  if src=0 and srclen<>0 then exit(r0=0)
  [struct+60h]=00000000h                ;\
  SHA1_Init(struct)                     ; first compute normal SHA1
  SHA1_Update(struct,src,srclen)        ; (same as SHA1_Init_Update_Finish)
  SHA1_Finish(first_sha1,struct)        ;/
 @@lop1:
  i=13h  ;start with LSB of big-endian 20-byte value  ;\increment SHA1 value
 @@lop2:                                              ; by one (with somewhat
  [first_sha1+i]=[first_sha1+i]+1, i=i-1               ; uncommon/bugged carry-
  if i>=0 and [first_sha1+i+1]=01h then goto @@lop2    ;/out to higher bytes)
  SHA1_Update(struct,first_sha1,14h)    ;\compute 2nd SHA1 across 1st SHA1,
  SHA1_Finish(second_sha1,struct)       ;/done without re-initializing struct
  for i=0 to min(14h,dstlen)-1, [dst]=[second_sha1+i], dst=dst+1
  dstlen=dstlen-min(14h,dstlen)
  if dstlen<>0 then goto @@lop1 else exit(r0=1)
```

SHA1_Init_Update_Finish_HMAC(dst,key,src,srclen)

```
  if len(key)>40h then key=SHA1(key) ;convert LONG keys to 14h-bytes length
  if len(key)<40h then zero-pad key to 40h-bytes length
  for i=0 to 3Fh, [inner_key+i]=[key+i] xor 36h ;\
  [struct+60h]=00000000h                        ;
  SHA1_Init(struct)                             ; compute 1st SHA1
  SHA1_Update(struct,inner_key,40h)             ; across inner key and data
  SHA1_Update(struct,src,srclen)                ;
  SHA1_Finish(first_sha1,struct)                ;/
  for i=0 to 3Fh, [outer_key+i]=[key+i] xor 5Ch ;\
  [struct+60h]=00000000h                        ;
  SHA1_Init(struct)                             ; compute final SHA1
  SHA1_Update(struct,outer_key,40h)             ; across outer key and 1st SHA1
  SHA1_Update(struct,first_sha1,14h)            ;
  SHA1_Finish(dst,struct)                       ;/
```

SWI 28h (DSi9/DSi7) - SHA1_Compare_20_Bytes(src1,src2)
Out: r0=1=match, r0=0=mismatch/error (error occurs if src1=0 or src2=0).

BIOS RSA Functions (DSi only)

RSA_Init_crypto_heap(heap_nfo,heap_start,heap_size)
RSA_Decrypt(heap_nfo,struct,dest4)
RSA_Decrypt_Unpad(heap_nfo,dst,src,key)
RSA_Decrypt_Unpad_OpenPGP_SHA1(heap_nfo,dst,src,key)

RSA is important because the DSi cartridge header and system files do contain RSA signatures. Which makes it impossible to create unlicensed software (without knowing Nintendo's private key).
BIOS RSA Basics
BIOS RSA Pseudo Code

SWI 20h (DSi9/DSi7) - RSA_Init_crypto_heap(heap_nfo,heap_start,heap_size)
Initializes the heap for use with SWI 21h..23h. heap_nfo is a 0Ch-byte structure, which gets set to:

```
  [heap_nfo+0] = heap_start (rounded-up to 4-byte boundary)
  [heap_nfo+4] = heap_end   (start+size, rounded-down to 4-byte boundary)
  [heap_nfo+8] = heap_size  (matched to above rounded values)
```

heap_start should point to a free memory block which will be used as heap, heap_size should be usually 1000h.

SWI 21h (DSi9/DSi7) - RSA_Decrypt(heap_nfo,ptr_nfo,len_dest)

```
  [ptr_nfo+0] = dst (usually 7Fh bytes, max 80h bytes)
  [ptr_nfo+4] = src (80h bytes)
  [ptr_nfo+8] = key (80h bytes)
```

This is a subfunction for SWI 22h/23h. It's returning raw decrypted data without unpadding (aside from stripping leading 00h bytes; most (or all) signatures are containing one leading 00h byte, so the returned [len_dest] value will be usually 7Fh).
Return value (r0) is: 0=failed, 1=okay. The length of the decrypted data is returned at [len_dest].

SWI 22h (DSi9/DSi7) - RSA_Decrypt_Unpad(heap_nfo,dst,src,key)
Same as SWI 21h, and additionally removes the "01h,min eight FFh,00h" padding. src,dst,key should be 80h-bytes. The output at dst can be theoretically max 80h-bytes (or shorter due to removed padding). In practice, the DSi is often using only the first 14h-bytes at dst (aka the last 14h-bytes from src) as SHA1 or SHA1-HMAC value (RSA-SHA1). Return value (r0) is: 0=failed, 1=okay.

SWI 23h (DSi9/DSi7) - RSA_Decrypt_Unpad_OpenPGP_SHA1(heap_nfo,dst,src,key)
Same as SWI 22h, but with some extra processing for extracting a SHA1 value from an OpenPGP header: The data must consist of five chunks (with IDs 30h,30h,06h,05h,04h), the last chunk (with ID=04h) must be 14h bytes in size, and the 14h-byte chunk data is then copied to dst. The other four chunks must exist, but their content is just skipped.

```
  00h 1    Leading zero       (00h)                          ;\
  01h 1    Block type         (01h)                          ; padding
  02h 5Ah  Padding Bytes      (FFh-filled)                   ;
  5Ch 1    Padding End        (00h)                          ;/
  5Dh 2    30h,junk(1)        (30h,21h)                      ;-whatever
  5Fh 2    30h,junk(1)        (30h,09h)                      ;-whatever
  61h 7    06h,len,junk(len)  (06h,05h, 2Bh,0Eh,03h,02h,1Ah) ;-OID for SHA1
  68h 2    05h,junk(1)        (05h,00h)                      ;-whatver
  6Ah 16h  04h,len,sha1(len)  (04h,14h, sha1[14h bytes])     ;-SHA1
```

The "junk" bytes aren't actually used/verified by the DSi. Handling chunks with len>7Fh looks quite weird/bugged. The DSi firmware contains some functions where it could optionally use SWI 22h or SWI 23h for RSA signatures (although, there aren't any known cases where it would actually use SWI 23h). Note: The DSi's Flipnote ".ppm" files are using the SWI 23h style SHA1 format (but Flipnote contains it's own RSA functions instead of using the BIOS SWI functions). DS Download Play (and equivalent code in NDS Firmware) is SWI 23h style, too (but also uses its own RSA function instead of the BIOS SWI).
Note: The format is based on the OpenPGP Message Format (RFC 4880), that format allows to use similar headers for MD5, SHA256, etc. (the DSi supports only SHA1 though).

Unencrypted Signatures
Emulators can hook the RSA BIOS functions to copy unencrypted signatures as-is (instead of trying to decrypt them). That allows to create "authentic" files that are fully compatible with the DSi firmware, and which would be actually working on real hardware (when knowing the private key).
Unencrypted 80h-byte signatures should be stored in following format (as defined in RFC 2313):

```
  00h   1      "00" Leading zero (00h)
  01h   1      "BT" Block type (always 01h on DSi)
  02h   8+n    "PS" Padding (FFh-filled, min 8 bytes, usually 69h bytes on DSi)
  0Ah+n 1      "00" Padding end (00h)
  0Bh+n 75h-n  "D"  Data (max 75h bytes, usually a 14h-byte SHA1 value on DSi)
```

If the hooked BIOS function sees a RSA source to contain "00h, 01h, at least eight FFh, followed by 00h", then it could treat it as unencrypted data (it's nearly impossible that an encrypted signature could contain those values).

Key.Bit0
The DSi BIOS contains two different RSA core modes (either one of them is used, depending on whether BIT0 of the FIRST BYTE of the "key" is set or cleared). The purpose & difference between those two modes is unknown. Also, dunno if that BIT0 thing is something common in the RSA world?

DSi Public RSA Keys (for verifying signatures)

```
  TWL_FIRM       (F1,F5,1A,FF..) eMMC Boot Info (same key for retail+debug)
  BIOS:FFFF87F4h (C3,02,93,DE..) Key0: System Menu (Launcher) of Retail version
  BIOS:FFFF8874h (B6,18,D8,61..) Key1: System Fun Tools and Wifi Firmware
```

```
BIOS:FFFF88F4h (DA,94,09,01..) Key2: System Base Tools (Settings, Shop)
BIOS:FFFF8974h (95,6F,79,0D..) Key3: DSiWare and DSi ROM Cartridges
BIOS:FFFF89F4h (D4,30,E3,7D..) Key4: Unknown ;\probably more/unused RSA keys
BIOS:FFFF8A74h (BD,29,02,38..) Key5: Unknown ; (DSi only)
BIOS:FFFF8AF4h (CF,8A,4B,15..) Key6: Unknown ; (doesn't exist on 3DS)
BIOS:FFFF8B74h (A3,BC,C1,7C..) Key7: Unknown ;/
BIOS:FFFF9920h (30,33,26,D5..) Unknown (probably NOT a RSA key)
Launcher       (BA,F1,98,A4..) HWINFO_S.dat (with RSA-SHA1-HMAC)
Launcher       (9F,80,BC,5F..) Version Data and TWLFontTable.dat
Launcher       (C7,F4,1D,27..) DS Cart Whitelist (missing RSA in v1.4E)
Launcher+NDS   (9E,C1,CC,C0..) For wifi-booted NDS titles (DsDownloadPlay)
Flipnote       (C2,3C,BC,13..) Public key for Flipnote .ppm files
Unknown        (?)             HWID.sgn
Unknown        (?)             Newer NDS ROM Cartridges (have RSA, too?)
DSi Shop       (9D,69,36,28..) Unknown, seems to be RSA        (100h bytes)
Launcher       (F8,24,6C,58..) Root key for cert.sys CA00000001(200h bytes)
cert.sys       (B2,79,C9,E2..) CA00000001 key for cert.sys keys(100h bytes)
cert.sys       (93,BC,0D,1F..) CP00000007 key for tmd's        (100h bytes)
cert.sys       (AD,07,A9,37..) XS00000003 key for shop-tickets (100h bytes)
cert.sys       (92,FF,96,40..) XS00000006 key for free-tickets (100h bytes)
cert.sys       (01,93,6D,08..) MS00000008 key for dev.kp       (ECC, non-RSA)
dev.kp         (per-console)   TWxxxxxxxx... key for tad files (ECC, non-RSA)
*.bin          (random?)       AP00030015484e42gg in tad files (ECC, non-RSA)
Launcher+Boot  (BC,FD,A1,FF..) Debug0: System Menu (Launcher, Debug version)
Launcher       (E9,9E,A7,9F..) Debug1:
Launcher       (A7,9F,54,A0..) Debug2:
Launcher       (AC,93,BB,3C..) Debug3: Public key for Debug DSiware/ROMs
Debug Updater  (E5,1C,BF,C7..) Debug Public key for HWInfo
Debug Updater  (C8,4B,38,2C..) Debug Public key for HWID.sgn    (100h bytes)
Launcher       (D0,1F,E1,00..) Debug Root key for CA00000002 key(200h bytes)
debug cert.sys (...)           Debug CA00000002 key for cert.sys(100h bytes)
debug cert.sys (...)           Debug CP00000005 key for ...?    (100h bytes)
debug cert.sys (...)           Debug CP00000007 key for ...     (100h bytes)
debug cert.sys (...)           Debug XS00000006 key for ...     (100h bytes)
verdata        (...)           Public keys in Version Data file?
Unknown        (?)             further keys...?
```

DSi Private RSA Keys (for creating signatures)

Nintendo's private keys should be known only by Nintendo. However, the console does have a few "own" private keys (for sending signed data to Nintendo (verdata), for storing signed data on SD card (flipnote), plus some more keys for the developer's debug version).

```
Flipnote       (26,A7,53,7E..) Private key for Flipnote .ppm files
dev.kp         (per-console)   TWxxxxxxxx... key for tad files (ECC, non-RSA)
(temp/unsaved?)(random?)       AP00030015484e42gg    tad files (ECC, non-RSA)
verdata        (...)           Private keys in Version Data file?
Debug Updater  (77,FC,77,9E..) Private key for Debug HWID.sgn (100h bytes)
Debug Updater  (B5,7C,C2,85..) Private key for Debug HWInfo
Debug SDK      (95,DC,C8,18..) Private key for Debug DSiware/ROMs (Debug3)
Unknown        (?)             further keys...?
```

The private keys are usually stored in DER format; that's including entries for the public key/exponent, and the original prime numbers that the key was generated from, plus some numbers that were temporarily used during key creation, and with all entries preceeded by tag/length values, whereas values with MSB set are preceeded by a 00h byte to make them "unsigned" (eg. most 80h-byte keys will occupy 81h bytes in DER format).

BIOS RSA Basics

RSA Basics

The RSA formulas are quite simple: Applying an exponent and modulus to the source data. There are two formulas used for encryption/decryption. The first formula requires only the Public Key (and an exponent, which is usually some fixed constant; on the DSi it's always 10001h aka 65537 decimal). The second formula is almost

same, but requires the Private Key instead of the constant exponent (and also requires the Public Key as modulus):

```
Public Key formula:      dest = src^10001h mod pubkey
Private Key formula:     dest = src^prvkey mod pubkey
```

That formulas can be used for encrypting secret messages, as so:

```
Recipient's Public Key   --> Encrypt a message
Recipient's Private Key  --> Decrypt a message
```

Or, using the formulas the other way around, to create digital signatures:

```
Sender's Private Key     --> Encrypt/create a signature
Sender's Public Key      --> Decrypt/verify a signature
```

The overall idea is that only the owner of the Private Key can decrypt messages, or create signatures. The Public Key can be shared freely, so that everybody can encrypt messages, or verify signatures.

RSA Big Number Maths

The exponent/modulus can be implemented with simple unsigned multiply/divide operations. However, RSA requires dealing with big 1024bit integers (or even bigger numbers when using larger keys), this does usually require some software functions since regular CPUs cannot directly deal with such large numbers.

RSA Byte Order

The DSi is storing all RSA keys and signatures in Big-Endian format, so one will need to reverse the byte order before doing the actual maths on Little-Endian CPUs.

RSA Signatures (used on DSi)

Digital signatures can be used for signing documents or other binaries. The signature does usually consist of a secure checksum (SHA-1, MD5, SHA256, etc.) computed on the document/binary, and then encrypted via the RSA Private Key formula.

The checksum can be then decrypted via Public Key, if the decrypted checksum does match up, then one can be sure that the document/binary hasn't been modified, and that it was really created by the Private Key owner.

RSA Encrypted Messages (not used on DSi)

Encrypted RSA messages are restricted to the size of the Public Key (eg. with a 1024bit key, the message should be smaller than 128 bytes). For bigger messages, one could either split the message into smaller snippets, or, one could combine RSA with some other encryption mechanism (eg. store an AES key in the RSA message, and decrypt the actual document via AES; that would add private/public key security to AES).

RSA Padding

RSA can be weak if the message is a small number (especially very small values like "0" or "1" obviously wouldn't work well with the "msg^exp" maths; other small values can be also weak, eg. with the common/small public exponent 10001h). To avoid that problem, the MSBs of the message should be padded with nonzero bytes, typically as so (as defined in RFC 2313):

```
00h   1      "00" Leading zero (00h)
01h   1      "BT" Block type (always 01h on DSi)
02h   8+n    "PS" Padding (FFh-filled, min 8 bytes, usually 69h bytes on DSi)
0Ah+n 1      "00" Padding end (00h)
0Bh+n 75h-n  "D"  Data (max 75h bytes, usually a 14h-byte SHA1 value on DSi)
```

That, for 80h-byte messages. For other sizes replace "75h" by "F5h, 1F5h, etc."

RSA Key Generation

Generating a RSA key pair is more difficult than the encryption/decryption part. First of, one needs two unsigned random prime numbers; for a 1024bit key, that would be usually two large 512bit prime numbers (whereas, finding real prime numbers is complicated, and it's more common to use values that have a "high probability" of being prime numbers).

The public key is then simply generated by multiplying the two prime numbers (P and Q) with each other:

```
pubkey = P * Q
```

The private key is also based on the same prime numbers, but the maths there are more complicated (and not described here).

When knowing one prime number, one could theoretically compute the other as "Q=pubkey/P", however, prime numbers aren't as rare as one might think, and it's quite impossible to guess (or brute-force) one of the prime numbers.

BIOS RSA Pseudo Code

rsa_mpi_pow_mod(dst,src,pubkey,exp,num_exp_bits) ;[dst]=[src]^[exp] mod [key]

```
  base(rsa__number_size), bigbuf(rsa_number_size*2)
  [base]=[src], [dst]=1, pow8bit=01h               ;-init base, result, powbit
  for i=1 to num_exp_bits
    if [exp] AND pow8bit then rsa_mpi_mul_mod(dst,base)   ;-mul result
    rsa_mpi_mul_mod(base,base)                            ;-square base
    pow8bit=pow8bit ROL 1, exp=exp+carry                  ;-next exp bit
  next i
  return
```

This is the RSA main function. The exponent is applied by squaring the "src" several times, and, if the corresponding exponent bit is set, multiplying the result by the squared value. To avoid the numbers to become incredible large, the modulus is applied after each multiplication (rather than applying it only on the final result).

```
  For the Private Key formula: Use exp=prvkey, num_exp_bits=rsa_number_size*8
  For the Public Key formula:  Use exp=ptr_to_10001h, num_exp_bits=17
```

The parameters and result for "rsa_mpi_pow_mod" must be in little-endian. Ie. for DSi, reverse the byte order of the incoming/outgoing values. And, on DSi, use rsa_number_size=80h (aka 128 bytes, aka for 1024bit RSA).

rsa_mpi_mul_mod(dst,src):

```
  rsa_mpi_mul(bigbuf,dst,src)          ;-multiply
  rsa_mpi_mod(bigbuf,pubkey)           ;-modulus
  [dst]=[bigbuf+0..rsa_number_size-1]  ;-copy to dst
  return
```

rsa_mpi_mul(dst,src1,src2): ;[dst]=[src1]*[src2]

```
  [dst+0]=0, oldmsw=0                  ;-init first word and oldmsw
  for i=0 to rsa_number_size-4 step 4  ;\
    call @@inner_loop                  ; compute LSWs of destination
    src2=src2+4                        ;
  next i                               ;/
  src2=src2-4
  for i=rsa_number_size-8 to 0 step -4 ;\
    src1=src1+4                        ; compute MSWs of destination
    call @@inner_loop                  ;
  next i                               ;/
  return
 ;---
 @@inner_loop:
  [dst+4]=oldmsw, oldmsw=0
  for j=0 to i step 4
    msw:lsw = [src1+j]*[src2-j]
    [dst+0]=[dst+0]+lsw
    [dst+4]=[dst+4]+msw+cy
    oldmsw=oldmsw+cy
  next j
  dst=dst+4
  ret
```

rsa_mpi_mod(dst,src): ;[dst]=[dst] mod [src] ;aka division remainder

;Double/Single -> Single modulo division (mpi/mpi)
;Divisor's MSW must be >= 80000000h

```
  ebx=rsa_number_size, dst=dst+ebx, i=ebx+4
 @@type0_lop:                                                    ;\
  if [dst+ebx-4]=0 then goto @@type0_next                        ;
```

```
 rsa_mpi_cmp(dst,src), if borrow then goto @@type1_next               ; type0
 rsa_mpi_sub(dst,src), if [dst+ebx-4]<>0 then goto @@type1_next       ; loop
@@type0_next:                                                        ;
 dst=dst-4, i=i-4, if i>0 then goto @@type0_lop                      ;/
 goto @@done
;--- --- ---
@@type1_lop:                                                         ;\
 lsw=[dst+ebx-4], msw=[dst+ebx-0]                                    ;
 if msw>=[src+ebx-4] then fac=FFFFFFFFh else fac=msw:lsw / [src+ebx-4] ;
 rsa_mpi_mulsub(dst,src,fac), if carry=0 then goto @@skip_add        ; type1
@@add_more:                                                          ; loop
 rsa_mpi_add(dst,src)                                                ;
 [dst+ebx]=[dst+ebx]+carry, if carry=0 then goto @@add_more          ;
@@skip_add:                                                          ;
 if [dst+ebx-4]=0 then goto @@type0_next                             ;
@@type1_next:                                                        ;
 dst=dst-4, i=i-4, if i>0 then goto @@type1_lop                      ;/
@@done:
 return
```

rsa_mpi_mulsub(dst,src,fac): ;[dst]=[dst]-[src]*fac

```
 oldborrow=0, oldmsw=0                                  ;\
 for i=0 to rsa_number_size-4 step 4                    ; process
   msw:lsw = [src+i]*fac, lsw=lsw+oldmsw, oldmsw=msw+carry  ; rsa_number_size
   [dst+i]=[dst+i]-lsw-oldborrow, oldborrow=borrow      ; bytes, plus...
 next i                                                 ;/
 [dst+rsa_number_size]=[dst+rsa_number_size]-oldmsw-oldborrow ;-one extra word
 return borrow   ;(unlike "rsa_embedded" which returns INVERTED borrow)
```

rsa_mpi_add(dst,src): ;out: [dst]=[dst]+[src], carry

```
 carry = 0
 for i=0 to rsa_number_size-4 step 4
   [dst+i]=[dst+i]+[src+i]+carry
 next i
 return carry
```

rsa_mpi_sub(dst,src): ;out: [dst]=[dst]-[src], borrow/unused

```
 borrow = 0
 for i=0 to rsa_number_size-4 step 4
   [dst+i]=[dst+i]-[src+i]-borrow
 next i
 return borrow
```

rsa_mpi_cmp[dst,src]: ;compare [dst]-[src], out: borrow

```
 for i=rsa_number_size-4 to 0 step -4
   temp=[dst+i]-[src+i], if not equal then return borrow
 next i
 return borrow
```

This is about same as "sub", but faster (because it can abort the loop upon first difference).

BIOS 3DS Exception Vectors

Exception Vectors

The 3DS bootroms do mostly contain bootcode, unlike GBA/NDS without any public functions, apart from some small stubs for redirecting the exception vectors to RAM locations:

```
 08000000h 8  arm9_irq               ldr r15,[$+4] // dd vector  ;\
 08000008h 8  arm9_fiq               ldr r15,[$+4] // dd vector  ;
 08000010h 8  arm9_svc               ldr r15,[$+4] // dd vector  ; ARM9
 08000018h 8  arm9_undef             ldr r15,[$+4] // dd vector  ; exceptions
 08000020h 8  arm9_prefetch_abort    ldr r15,[$+4] // dd vector  ;
```

```
08000028h 8  arm9_data_abort         ldr r15,[$+4] // dd vector  ;/
1FFFFFA0h 8  arm11_irq               ldr r15,[$+4] // dd vector  ;\
1FFFFFA8h 8  arm11_fiq               ldr r15,[$+4] // dd vector  ;
1FFFFFB0h 8  arm11_svc               ldr r15,[$+4] // dd vector  ; ARM11
1FFFFFB8h 8  arm11_undef             ldr r15,[$+4] // dd vector  ; exceptions
1FFFFFC0h 8  arm11_prefetch_abort    ldr r15,[$+4] // dd vector  ;
1FFFFFC8h 8  arm11_data_abort        ldr r15,[$+4] // dd vector  ;/
1FFFFFDCh 4  arm11_core1_entrypoint  dd vector                   ;-CPU1 entry
(OVERLAY) 4  arm11_core23_entry      dd vector                   ;-CPU2/3
```

The ARM11 exception vectors are shared for all CPU cores (so one must manually redirect them to core specific handlers).
Alternately, instead of using the above RAM vectors, one could use ARM11 virtual memory or ARM9 ITCM for custom vectors.
Moreover, the CFG11_BOOTROM_OVERLAY_CNT/VAL feature allows to redirect ARM11 vectors on New3DS (mainly intended for booting CPU2/CPU3 cores).

SWI/SVC Functions
There aren't any built-in functions in the bootroms, however, SVC opcodes can be used to obtain entrypoints to OS functions. That is, once when the OS is booted (FIRM files are started without any OS functions, but .code files can use OS functions).
Note: ARM has renamed SWI opcodes to SVC opcodes.

BIOS RAM Usage

Below contains info about RAM contents at cartridge boot time (as initialized by the BIOS/Firmware), plus info about RAM locations used by IRQ handlers and SWI functions.

GBA BIOS RAM Usage
Below memory at 3007Fxxh is often accessed directly, or via mirrors at 3FFFFxxh.

```
3000000h 7F00h User Memory and User Stack              (sp_usr=3007F00h)
3007F00h A0h   Default Interrupt Stack (6 words/time)  (sp_irq=3007FA0h)
3007FA0h 40h   Default Supervisor Stack (4 words/time) (sp_svc=3007FE0h)
3007FE0h 10h   Debug Exception Stack (4 words/time)    (sp_xxx=3007FF0h)
3007FF0h 4     Pointer to Sound Buffer (for SWI Sound functions)
3007FF4h 3     Reserved (unused)
3007FF7h 1     Reserved (intro/nintendo logo related)
3007FF8h 2     IRQ IF Check Flags (for SWI IntrWait/VBlankIntrWait functions)
3007FFAh 1     Soft Reset Re-entry Flag (for SWI SoftReset function)
3007FFBh 1     Reserved (intro/multiboot slave related)
3007FFCh 4     Pointer to user IRQ handler (to 32bit ARM code)
```

NDS BIOS RAM Usage
Below memory at 27FFxxxh is mirrored to 23FFxxxh (on retail consoles with 4MB RAM), however, it should be accessed via address 27FFxxxh (for compatibility with debug consoles with 8MB RAM). Accessing it via mirrors at 2FFFxxxh is also valid (this is done by DSi enhanced games; even when running in non-DSi mode; this allows DSi games to use the same memory addresses in NDS and DSi mode).

```
2000000h ...   ARM7 and ARM9 bootcode can be loaded here (2000000h..23BFDFFh)
2400000h ...   Debug bootcode can be loaded here (2400000h..27BFDFFh)
23FEE00h 168h  Fragments of NDS9 firmware boot code
27FF800h 4     NDS Gamecart Chip ID 1
27FF804h 4     NDS Gamecart Chip ID 2
27FF808h 2     NDS Cart Header CRC (verified)           ;hdr[15Eh]
27FF80Ah 2     NDS Cart Secure Area CRC (not verified ?) ;hdr[06Ch]
27FF80Ch 2     NDS Cart Missing/Bad CRC (0=Okay, 1=Missing/Bad)
27FF80Eh 2     NDS Cart Secure Area Bad (0=Okay, 1=Bad)
27FF810h 2     Boot handler task number (usually FFFFh at cart boot time)
27FF812h 2     Secure disable (0=Normal, 1=Disable; Cart[078h]=BIOS[1088h])
27FF814h 2     SIO Debug Connection Exists (0=No, 1=Yes)
```

```
27FF816h 2     RTC Status?                 (0=Okay, 1=Bad)
27FF818h 1     Random RTC    ;random LSB from SIO debug detect handshake
27FF819h 37h   Zerofilled by firmware
27FF850h 2     NDS7 BIOS CRC (5835h)
27FF860h 4     Somewhat copy of Cart[038h], nds7 ram addr (?)
27FF864h 4     Wifi FLASH User Settings Bad (0=Okay, 1=Bad)
27FF868h 4     Wifi FLASH User Settings FLASH Address (fmw[20h]*8)
                 maybe recommended to use above RAM cell instead FLASH entry?
27FF86Ch 4     Whatever (seems to be zero at cart boot time)
27FF870h 4     Whatever (seems to be zero at cart boot time)
27FF874h 2     Wifi FLASH firmware part5 crc16 (359Ah) (fmw[026h])
27FF876h 2     Wifi FLASH firmware part3/part4 crc16 (fmw[004h] or ZERO)
                 Above is usually ZERO at cart boot (set to fmw[004h] only
                 when running pictochat, or maybe also when changing user
                 settings)
27FF878h 08h   Not used
27FF880h 4     Message from NDS9 to NDS7  (=7 at cart boot time)
27FF884h 4     NDS7 Boot Task (also checked by NDS9) (=6 at cart boot time)
27FF888h ..    Whatever (seems to be zero at cart boot time)
27FF890h 4     Somewhat boot flags (somewhat B0002A22h)
                 bit10 part3/part4 loaded/decoded (bit3 set if bad crc)
                 bit28 part5 loaded/decoded with good crc
27FF894h 36Ch  Not used (zero)
27FFC00h 4     NDS Gamecart Chip ID 1   (copy of 27FF800h)
27FFC04h 4     NDS Gamecart Chip ID 2   (copy of 27FF804h)
27FFC08h 2     NDS Cart Header CRC      (copy of 27FF808h)
27FFC0Ah 2     NDS Cart Secure Area CRC (copy of 27FF80Ah)
27FFC0Ch 2     NDS Cart Missing/Bad CRC (copy of 27FF80Ch)
27FFC0Eh 2     NDS Cart Secure Area Bad (copy of 27FF80Eh)
27FFC10h 2     NDS7 BIOS CRC (5835h)    (copy of <27FF850h>)
27FFC12h 2     Secure Disable           (copy of 27FF812h)
27FFC14h 2     SIO Debug Exist          (copy of 27FF814h)
27FFC16h 1     RTC Status?              (<8bit> copy of 27FF816h)
27FFC17h 1     Random 8bit              (copy of <27FF818h>)
27FFC18h 18h   Not used (zero)
27FFC30h 2     GBA Cartridge Header[BEh], Reserved
27FFC32h 3     GBA Cartridge Header[B5h..B7h], Reserved
27FFC35h 1     Whatever flags ?
27FFC36h 2     GBA Cartridge Header[B0h], Maker Code
27FFC38h 4     GBA Cartridge Header[ACh], Gamecode
27FFC3Ch 4     Frame Counter (eg. 00000332h in no$gba with original firmware)
27FFC40h 2     Boot Indicator (0001h=normal; required for some NDS games)
27FFC42h 3Eh   Not used (zero)
27FFC80h 70h   Wifi FLASH User Settings (fmw[newest_user_settings])
27FFCF0h 10h   Not used (zero)
27FFDxxh ..    NDS9 Debug Exception Stack (stacktop=27FFD9Ch)
27FFD9Ch 4     NDS9 Debug Exception Vector (0=None)
27FFDA0h ..    ...
27FFE00h 170h  NDS Cart Header at 27FFE00h+0..16Fh
27FFF70h ..    Not used (zerofilled at cart boot time)
27FFFF8h 2     NDS9 Scratch addr for SWI IsDebugger check
27FFFFAh 2     NDS7 Scratch addr for SWI IsDebugger check
27FFFFCh ..    ...
27FFFFEh 2     Main Memory Control (on-chip power-down I/O port)
DTCM+3FF8h 4   NDS9 IRQ IF Check Bits (hardcoded RAM address)
DTCM+3FFCh 4   NDS9 IRQ Handler (hardcoded RAM address)
37F8000h FE00h ARM7 bootcode can be loaded here (37F8000h..3807DFFh)
380F700h 1D4h  Fragments of NDS7 firmware boot code
380F980h 4     Unknown/garbage (set to FBDD37BBh, purpose unknown)
                 NOTE: Cooking Coach is doing similar crap at 37FCF1Ch ?!?!
380FFC0h 4     DSi7 IRQ IF2 Check Bits (hardcoded RAM address) (DSi only)
380FFDCh ..    NDS7 Debug Stacktop / Debug Vector (0=None)
380FFF8h 4     NDS7 IRQ IF Check Bits (hardcoded RAM address)
380FFFCh 4     NDS7 IRQ Handler (hardcoded RAM address)
---
```

```
summary of nds memory used at cartridge boot time:
(all other memory zero-filled unless containing cartridge data)
37F8000h..3807E00h  ;cartridge area (nds7 only)
2000000h..23BFE00h  ;cartridge area (nds9 and nds7)
2400000h..27BFE00h  ;cartridge area (debug ver)
23FEE00h..23FEF68h  ;fragments of NDS9 firmware boot code
27FF800h..27FF85Fh  ;various values (from BIOS boot code)
27FF860h..27FF893h  ;various values (from Firmware boot code)
27FFC00h..27FFC41h  ;various values (from Firmware boot code)
27FFC80h..27FFCE6h  ;firmware user settings
27FFE00h..27FFF6Fh  ;cart header
380F700h..380F8D4h  ;fragments of NDS7 firmware boot code
380F980h            ;set to FBDD37BBh
---
register settings at cartridge boot time:
nds9 r0..r11     = zero
nds9 r12,r14,r15 = entrypoint
nds9 r13         = 3002F7Ch (!)
nds9 r13_irq     = 3003F80h
nds9 r13_svc     = 3003FC0h
nds9 r14/spsr_irq= zero
nds9 r14/spsr_svc= zero
---
nds7 r0..r11     = zero
nds7 r12,r14,r15 = entrypoint
nds7 r13         = 380FD80h
nds7 r13_irq     = 380FF80h
nds7 r13_svc     = 380FFC0h
nds7 r14/spsr_irq= zero
nds7 r14/spsr_svc= zero
---
Observe that SWI SoftReset applies different stack pointers:
Host  sp_svc    sp_irq    sp_sys    zerofilled area       return address
NDS7  380FFDCh  380FFB0h  380FF00h  [380FE00h..380FFFFh]  Addr[27FFE34h]
NDS9  0803FC0h  0803FA0h  0803EC0h  [DTCM+3E00h..3FFFh]   Addr[27FFE24h]
```

DSi BIOS RAM

```
2000000h 8     AutoParam Old Title ID (former title)    ;carthdr[230h]
2000008h 1     AutoParam Unknown/Unused
2000009h 1     AutoParam Flags (03h=Stuff is used?)
200000Ah 2     AutoParam Old Maker code                 ;carthdr[010h]
200000Ch 2     AutoParam Unknown (02ECh) ;\counter/length/indices/whatever?
200000Eh 2     AutoParam Unknown (0000h) ;/
2000010h 2     AutoParam CRC16 on [000h..2FFh], initial=FFFFh, [010h]=0000h
2000012h 2     AutoParam Unknown/Unused (000Fh = want Internet Settings?)
2000014h 2ECh  AutoParam Unknown... some buffer... string maybe?
2000300h 4     AutoLoad ID ("TLNC") (also requires BPTWL[70h]=01h)
2000304h 1     AutoLoad Unknown/unused (usually 01h)
2000305h 1     AutoLoad Leng of data at 2000308h (01h..18h,for CRC,18h=norm)
2000306h 2     AutoLoad CRC16 of data at 2000308h (with initial value FFFFh)
2000308h 8     AutoLoad Old Title ID (former title) (can be 0=anonymous)
2000310h 8     AutoLoad New Title ID (new title to be started, 0=none)
2000318h 4     AutoLoad Flags (bit0,1-3,4,5,6,7) ;usually 16bit, once 32bit
200031Ch 4     AutoLoad Unused (but within checksummed area when CRC len=18h)
2000320h E0h   AutoLoad Unused (but zerofilled upon erasing autoload area)
2000400h 128h  System Settings from TWLCFGn.dat file (bytes 088h..1AFh)
20005E0h 1     WlFirm Type (1=DWM-W015, 2=DWM-W024) (as wifi_flash[1FDh])
20005E1h 1     WlFirm Unknown (zero)
20005E2h 2     WlFirm CRC16 with initial value FFFFh on [20005E4h..20005EFh]
20005E4h 4     WlFirm RAM vars (500400h)  ;\
20005E8h 4     WlFirm RAM base (500000h)  ; as from "Wifi Firmware" file
20005ECh 4     WlFirm RAM size (02E000h)  ;/
20005F0h 10h   WlFirm Unknown (zero)
2000600h 14h   Hexvalues from HWINFO_N.dat
```

```
2000800h ...   Unlaunch Auto-load feature (via "device:/Path/Filename.ext")
23FEE00h 200h  DSi9 bootstrap relict
---
2FEE120h 4     "nand"   <--- passed as so to launcher
2FF80xxh
2FF82xxh
2FF83xxh
2FF89xxh
2FF8Axxh
2FF8Bxxh
2FF8Cxxh
2FF8Dxxh       ... Wifi MAC address, channel mask, etc.
2FF8Fxxh
2FF90xxh
2FF91xxh
2FF9208h       FBDD37BBh (that odd "garbage" value occurs also on NDS)
2FFA1xxh
2FFA2xxh
2FFA5xxh
2FFA6xxh
2FFA680h 12    02FD4D80h,00000000h,00001980h
2FFA68Ch ..    Zerofilled
---
2FFC000h 1000h Full Cart Header (as at 2FFE000h, but, FOR NDS ROM CARTRIDGE)
2FFD000h 7B0h  Zerofilled
2FFD7B0h 8+1   Version Data Filename (eg. 30,30,30,30,30,30,30,34,00)
2FFD7B9h 1     Version Data Region   (eg. 50h="P"=Europe)
2FFD7BAh 1     Unknown  (00)   ;bit0 = warmboot-flag-related
2FFD7BBh 1     Unknown  (00)
2FFD7BCh 15+1  eMMC CID (dd,ss,ss,ss,ss,03,4D,30,30,46,50,41,00,00,15), 00
2FFD7CCh 15+1  eMMC CSD (40,40,96,E9,7F,DB,F6,DF,01,59,0F,2A,01,26,90), 00
2FFD7DCh 4     eMMC OCR (80,80,FF,80)                                  ;20h
2FFD7E0h 8     eMMC SCR (00,04,00,00,00,00,00,00) (for MMC: dummy/4bit);24h
2FFD7E8h 2     eMMC RCA (01,00)                                        ;2Ch
2FFD7EAh 2     eMMC Typ (01,00) (0=SD Card, 1=MMC Card)                ;2Eh
2FFD7ECh 2     eMMC HCS (00,00) ;copy of OCR.bit30 (sector addressing) ;30h
2FFD7EEh 2     eMMC ?   (00,00)                                        ;32h
2FFD7F0h 4     eMMC ?   (00,00,00,00)                                  ;34h
2FFD7F4h 4     eMMC CSR (00,09,00,00)   ;card status (state=tran)      ;38h
2FFD7F8h 2     eMMC Port 4004824h setting  (00,01)  ;SD_CARD_CLK_CTL   ;3Ch
2FFD7FAh 2     eMMC Port 4004828h setting  (E0,40)  ;SD_CARD_OPTION    ;3Eh
2FFD7FCh 2     eMMC ?   (00,00)                                        ;40h
2FFD7FEh 2     eMMC Device (usually 0001h=eMMC) (0000h=SD/MMC Slot?)   ;42h
2FFD800h 1     Titles: Number of titles in below lists (max 76h)
2FFD801h 0Fh   Titles: Zerofilled
2FFD810h 10h   Titles: Pub Flags (1bit each) ;same maker plus public.sav
2FFD820h 10h   Titles: Prv Flags (1bit each) ;same maker plus private.sav
2FFD830h 10h   Titles: Jmp Flags (1bit each) ;jumpable or current-title
2FFD840h 10h   Titles: Mkr Flags (1bit each) ;same maker
2FFD850h 3B0h  Titles: Title IDs (8 bytes each)
2FFDC00h 400h  Zerofilled
2FFE000h 1000h DSi Full Cart Header (additionally to short headers)
2FFF000h 0Ch   Zerofilled
2FFF00Ch 4     ? 0000007Fh
2FFF010h 4     ? 550E25B8h
2FFF014h 4     ? 02FF4000h
2FFF018h A68h  Zerofilled
2FFFA80h 160h  Short Cart header (as at 2FFFE00h, but, FOR NDS ROM CARTRIDGE)
2FFFBE0h 20h   Zerofilled
```

Below resembles NDS area at 27FFC00h (with added/removed stuff)...

```
2FFFC00h 4     NDS Gamecart Chip ID
2FFFC04h 20h   Zerofilled
2FFFC24h 5     ? (04 00 73 01 03)
2FFFC29h 7     Zerofilled
2FFFC30h 12    GBA Cartridge Header (FF FF FF FF FF 00 FF FF FF FF FF FF)
```

```
 2FFFC3Ch 4     Frame Counter maybe? (eg. 1F 01 00 00 in cooking coach)
 2FFFC40h 2     Boot Indicator (1=ROM Cartridge,2=Wifi/tmp?,3=SD/MMC DSiware)
 2FFFC42h 3Eh   Not used (zero)
 2FFFC80h 70h   Wifi FLASH User Settings (fmw[newest_user_settings])
 2FFFCF0h 4     ?           (3D 00 01 6E) (update counter and crc16 ?)
 2FFFCF4h 6     Wifi MAC Address (00 23 CC xx xx xx)              (fmw[036h])
 2FFFCFAh 2     Wifi Channels (usually 1041h = ch1+7+13) (based on fmw[03Ch])
 2FFFCFCh 4     Zero
 2FFFD00h 68h   Zerofilled
 2FFFD68h 4     Bitmask for Supported Languages (3Eh for Europe);\
 2FFFD6Ch 4     Unknown (00,00,00,00)                           ; from
 2FFFD70h 1     Console Region (0=JP,1=US,2=EU,3=AU,4=CHN,5=KOR); HWINFO_S.dat
 2FFFD71h 12    Serial/Barcode (ASCII, 11-12 characters)        ;
 2FFFD7Dh 3     ? (00 00 3C)                                    ;/
 2FFFD80h 0Ch   Zerofilled
 2FFFD8Ch 10h   ARM9 debug exception stack (stacktop 2FFFD9Ch)
 2FFFD9Ch 4     ARM9 debug exception vector (020D3E64h)
 2FFFDA0h 4     02F80000h                                  ;\
 2FFFDA4h 4     02FFA674h                                  ;
 2FFFDA8h 4     00000000h zero                             ; start addresses?
 2FFFDACh 4     01FF86E0h itcm?                            ;
 2FFFDB0h 4     027C00C0h                                  ;
 2FFFDB4h 4     02FFF000h                                  ;
 2FFFDB8h 4     03040000h wram?                            ;
 2FFFDBCh 4     03800000h wram?                            ;
 2FFFDC0h 4     0380C3B4h wram?                            ;/
 2FFFDC4h 4     02F80000h                                  ;\
 2FFFDC8h 4     02FFC000h ptr to DSi Full Cart Header      ;
 2FFFDCCh 4     00000000h zero                             ; end addresses?
 2FFFDD0h 4     02000000h ram bottom?                      ; (for above nine
 2FFFDD4h 4     027C0780h                                  ; start addresses)
 2FFFDD8h 4     02FFF680h                                  ;
 2FFFDDCh 4     03040000h wram?                            ;
 2FFFDE0h 4     03800000h wram?                            ;
 2FFFDE4h 4     0380F780h wram?                            ;/
 2FFFDE8h 4     RTC Date at Boot (BCD) (yy,mm,dd,XX) (XX=maybe day-of-week?)
 2FFFDECh 4     RTC Time at Boot (BCD) (hh,ss,mm,0) (hh.bit6=maybe PM or 24h?)
 2FFFDF0h 4     Initial ARM7 Port 4004008h bits (13FBFB06h) (SCFG_EXT)
 2FFFDF4h 1     Initial ARM7 Port 40040xxh bits (C4h)       (SCFG_xxx)
 2FFFDF5h 1     Initial ARM7 Port 400400xh bits (F0h)       (SCFG_xxx)
 2FFFDF6h 2+2   Zerofilled
 2FFFDFAh 1     Warmboot Flag (bptwl[70h] OR 80h, ie. 80h=cold or 81h=warm)
 2FFFDFBh 1     01h
 2FFFDFCh 4     Pointer to TWLCFGn.dat (usually 2000400h) (or 0=2000400h)
 2FFFE00h 160h  Short Cart header (unlike NDS, only 160h, not 170h)
 2FFFF60h A0h   Zerofilled
37FA414h       "nand:/title/....app"  <-- [1D4h]+3C0h  (without Device List!)
380C400h 22E4h BIOS Keys (as from Boot Stage 1, see there)
 380F010h 10h   AES key for dev.kp (E5,CC,5A,8B,...) (optional/for launcher)
 380F600h 200h  DSi7 bootstrap relict (at 3FFF600h aka mirrored to 380F600h)
 380FFC0h 4     DSi7 IRQ IF2 Check Bits (hardcoded RAM address) (DSi only)
 380FFC4h 4     DSi7 SCFG_EXT setting
 380FFC8h 2     DSi7 SCFG_misc bits
 380FFDCh ..    DSi7 Debug Stacktop / Debug Vector (0=None)
 380FFF8h 4     DSi7 IRQ IF Check Bits (hardcoded RAM address)
 380FFFCh 4     DSi7 IRQ Handler (hardcoded RAM address)
 xxxxxxxh ?     ARM7i and ARM9 bootcode can be loaded WHERE and WHERE?
 cart_header[1D4h] 400h SD/MMC Device List ARM7 RAM; initialized by firmware
```

Initial state after DSi BIOS ROM bootcode (when starting eMMC bootcode) requires only a few memory blocks in ITCM, ARM7 WRAM, and AES keyslots:

```
 1FFC400h 400h  BIOS Keys from FFFF87F4h (C3 02 93 DE ..) RSA keys (8x80h)
 1FFC800h 80h   BIOS Keys from FFFF9920h (30 33 26 D5 ..) Whatever
 1FFC880h 14h   Whatever, should/may be zerofilled?
 1FFC894h 1048h BIOS Keys from FFFF99A0h (99 D5 20 5F ..) Blowfish/NDS-mode
```

```
1FFD8DCh 1048h BIOS Keys from FFFFA9E8h (D8 18 FA BF ..) Blowfish/unused?
3FFC400h 200h  BIOS Keys from 00008188h (CA 13 31 79 ..) Whatever, 32x10h AES?
3FFC600h 40h   BIOS Keys from 0000B5D8h (AF 1B F5 16 ..) Whatever, AES?
3FFC640h 14h   Whatever, must be zerofilled
3FFC654h 1048h BIOS Keys from 0000C6D0h (59 AA 56 8E ..) Blowfish/DSi-mode
3FFD69Ch 1048h BIOS Keys from 0000D718h (54 86 13 3B ..) Blowfish/unused?
3FFE6E4h 44h   eMMC Info (to be relocated to 2FFD7BCh, see there for details)
4004450h 8     AES Key0.X ("Nintendo") for modcrypt
4004480h 10h   AES Key1.X (CPU/Console ID and constants) for dev.kp and Tad
40044B0h 10h   AES Key2.X ("Nintendo DS",...)            for Tad
40044E0h 1Ch   AES Key3.X/Y (CPU/Console ID and constants) for eMMC
2000000h ...   Warmboot Param (optional, passed on to New Title)
2000300h 20h   Warmboot Info  (optional, passed on to Launcher)
```

BIOS Dumping

BIOSes

```
GBA BIOS 16K  (fully dumpable, CRC32=81977335 on GBA with [3F0Ch]=00h)
GBA BIOS 16K  (fully dumpable, CRC32=A6473709 on NDS/3DS with [3F0Ch]=01h)
NDS7 BIOS 16K (fully dumpable, CRC32=1280F0D5)
NDS9 BIOS 4K  (fully dumpable, CRC32=2AB23573)
DSi7 BIOS 64K (about 41K dumpable)
DSi9 BIOS 64K (about 41K dumpable)
3DS9 BIOS 64K (fully dumpable)
3DS11 BIOS 64K (fully dumpable)
DSiWifi BIOS 80K on older DSi (fully dumpable)
DSiWifi BIOS Unknown size on newer DSi (fully dumpable)
3DSWifi BIOS Unknown size on 3DS (fully dumpable)
```

GBA BIOS

Contains SWI Functions and Bootcode (for starting cartridges, or booting via Serial Port). The GBA BIOS can be read only by opcodes executed in BIOS area, for example, via the MidiKey2Freq function (most other SWI Functions (like CpuSet) are refusing source addresses within BIOS area).

NDS BIOSes

Contains SWI Functions and Bootcode (for booting from SPI Bus Firmware FLASH memory). The NDS9 BIOS can be dumped without restrictions (eg. via CpuSet, or via LDR opcodes in RAM). The NDS7 BIOS has same restrictions as GBA, ie. reading works only by BIOS opcodes, and not by functions like CpuSet. The GetCRC16 functions does work though (at least for memory at 1204h..3FFFh). As an additional obstacle, memory at 0000h..1203h can be dumped only by opcodes within 0000h..1203h (that memory does mainly contain data, but some of the data values can serve as THUMB LDR opcodes). For details see:

DS Memory Control - BIOS

Note: DSi consoles are containing a copy of the NDS BIOSes, but with BIOSPROT set to 0020h (even when running in NDS mode), so the first 20h bytes of the DSi's NDS7 BIOS aren't dumpable (except via tracing, see below), that 20h bytes should be just same as on original NDS7 though.

DSi BIOSes - Lower 32K-halves (SWI Functions)

The lower 32K of DSi9 doesn't have any restricions. The lower 32K of DSi7 has similar restrictions as NDS7, but with BIOSPROT set to 0020h (instead of 1204h), this is making it more easy to dump memory at 0020h..7FFFh (eg. via GetCRC16), but makes it impossible to dump the exception vectors at 0000h..001Fh, however, they can be deduced by tracing (with timer IRQs):

```
ROM:00000000h  EA000006  b 20h   ;dsi7_reset_vector
ROM:00000004h  EA000006  b 24h   ;dsi7_undef_handler
ROM:00000008h  EA00001F  b 8Ch   ;dsi7_swi_handler
ROM:0000000Ch  EA000004  b 24h   ;dsi7_prefetch_abort_handler
ROM:00000010h  EA000003  b 24h   ;dsi7_data_abort_handler
ROM:00000014h  EAFFFFFE  b 14h   ;reserved_vector
```

```
ROM:00000018h  EA000013  b 6Ch   ;dsi7_irq_handler
ROM:0000001Ch  EA000000  b 24h   ;dsi7_fiq_handler
```
Aside from branch opcodes, above could theoretically contain ALU opcodes that modify R15 (but that would be very unlikely, and would make no difference) (and, the above 20h bytes are dumped/confirmed on 3DS in DSi mode).

DSi BIOSes - Upper 32K-halves (Bootcode, for booting from eMMC memory)

The upper 32K of the DSi9 and DSi7 BIOSes are locked at some point during booting, and there's no known way to dump them directly. However, portions of that memory are relocated to RAM/TCM before locking, and that relocated copies can be dumped.

On a DSi, the following DSi ROM data can be dumped (originally done via Main Memory hacks, ie. with complex external hardware soldered to the mainboard, but it's now also possible via Unlaunch.dsi exploit):
```
ROM:FFFF87F4h / TCM:1FFC400h (400h)  (C3 02 93 DE ..) RSA keys (8x80h)
ROM:FFFF9920h / TCM:1FFC800h (80h)   (30 33 26 D5 ..) Whatever
ROM:FFFF99A0h / TCM:1FFC894h (1048h) (99 D5 20 5F ..) Blowfish/NDS-mode
ROM:FFFFA9E8h / TCM:1FFD8DCh (1048h) (D8 18 FA BF ..) Blowfish/unused?
ROM:00008188h / RAM:3FFC400h (200h)  (CA 13 31 79 ..) Whatever, 32x10h AES?
ROM:0000B5D8h / RAM:3FFC600h (40h)   (AF 1B F5 16 ..) Whatever, "common key"?
ROM:0000C6D0h / RAM:3FFC654h (1048h) (59 AA 56 8E ..) Blowfish/DSi-mode
ROM:0000D718h / RAM:3FFD69Ch (1048h) (54 86 13 3B ..) Blowfish/unused?
```
On a 3DS, the following "DSi ROM data" can be dumped from the 2470h-byte DSi key area in 3DS memory at ARM9 ITCM 01FFD000h..01FFF46F (via 3DS exploits that are capable of executing code on ARM9 side):
```
ROM:FFFF87F4h / 3DS:01FFD000h  200h  RSA keys 0..3 (4x80h)
ROM:00008308h / 3DS:01FFD200h  80h   some AES keys
ROM:FFFF9920h / 3DS:01FFD280h  80h   whatever
ROM:0000B5D8h / 3DS:01FFD300h  40h   AES keys and values (common etc)
ROM:?         / 3DS:01FFD340h  A0h   misc "Nintendo" string etc.
ROM:0000C6D0h / 3DS:01FFD3E0h  1048h Blowfish for DSi-mode
ROM:FFFF99A0h / 3DS:01FFE428h  1048h Blowfish for DS-mode
```
The 3DS does have only half of the DSi keys (the extra keys might be used for DSi debug version, but aren't needed for normal DSi software).

The 40h-byte area for ROM:0000B5D8h can be fully dumped from 3DS ITCM, the same vales should also exist in DSi ITCM, but the DSi zerofills a 10h-byte fraction of that area after initialization, and it doesn't seem be possible to read that values via Main Memory hacks (most of that erased values can be found in AES keyslots though).

The A0h-byte area is found only in 3DS ITCM, it should also exist somewhere in DSi ROM, but isn't relocated to DSi ITCM (however, the relevant values can be found in AES keyslots, eg. the "Nintendo" string).

Checksums for BiosDSi.rom (20000h bytes)
```
Offset Size  CRC32
00000h 8000h 5434691Dh               ;\
08000h 188h  ?                       ;
08188h 180h  E5632151h  (not 3ds)    ;
08308h 80h   64515306h               ;
08388h 3250h ?                       ;
0B5D8h 20h   85BE2749h               ; ARM7
0B5F8h 10h   25A46A54h  (3ds only)   ;
0B608h 10h   E882B9A9h               ;
0B618h 10B8h ?                       ;
0C6D0h 1048h 3B5CDF06h               ;
0D718h 1048h 5AC363F9h  (not 3ds)    ;
0E860h 18A0h ?                       ;/
10000h 8000h 11E7C1EAh               ;\
18000h 7F4h  ?                       ;
187F4h 200h  4405D4BAh               ;
189F4h 200h  2A32F2E7h  (not 3ds)    ;
18BF4h D2Ch  ?                       ; ARM9
19920h 80h   2699A10Fh               ;
199A0h 1048h A8F58AE7h               ;
1A9E8h 1048h E94759ACh  (not 3ds)    ;
```

```
  1BA30h 45D0h ?                         ;/
  ?      A0h   180DF59Bh  (3ds only)    ;-whatever, "Nintendo" string etc.
  ?      80h   ........h  (TWL-FIRM)    ;-RSA key for eMMC boot info
```
Checksums for the 'whole' 20000h-byte file (with unknown/missing areas zerofilled):
```
  180DF59Bh (tcm/ram dump)  (missing 10h bytes)
  03A21235h (3ds dump)      (missing 180h+200h+1048h+1048h bytes)
  CDAA8FF6h (combined dump) (missing only the unknown "?" areas)
```

3DS BIOS
Can be dumped by crashing the 3DS with wrong voltages, or by using sighax to crash the bootrom before disabling the upper 32Kbyte halves of the ARM9/ARM11 ROMs.

DSiWifi BIOS
The Wifi BIOS can be dumped by using the WINDOW_DATA register via SDIO CMD53.

Further DSi BIOSes
The DSi cameras and several other I2C/SPI devices are probably having BIOS ROMs, too. Unknown if/how that ROMs are dumpable.

DSi BIOS Dumping via voltage errors
Lowering VDD12 for a moment does work quite reliable for crashing the ARM9 and trapping the 2FFFD9Ch vector in Main RAM. The problem is that Main RAM seems to be disabled during bootstage 1 (it gets enabled at begin of bootstage 2 via EXMEMCNT, that is, shortly after the upper BIOS 32Kbyte areas are disabled). More on that here:
```
  http://4dsdev.kuribo64.net/thread.php?id=130
```
One theory/idea (from dark_samus) is that EXMEMCNT controls the CE2 pin on the Main RAM chip, so one could try to rewire that pin to get Main RAM enabled regardless of EXMEMCNT, if that's actually working, then trapping the 2FFFD9Ch vector should work even while BIOS ROMs are fully readable.

External Connectors

External Connectors
AUX GBA Game Pak Bus
AUX DS Game Card Slot
AUX Link Port
AUX Sound/Headphone Socket and Battery/Power Supply
AUX DSi SD/MMC Pin-Outs

Getting access to Internal Pins
AUX Opening the GBA
AUX Mainboard
AUX DSi Component Lists
AUX DSi Internal Connectors
AUX DSi Chipset Pinouts

More Internal Stuff
Pinouts - CPU - Signal Summary
Pinouts - CPU - Pinouts
Pinouts - Audio Amplifiers
Pinouts - LCD Cables
Pinouts - Power Switches, DC/DC Converters, Reset Generators
Pinouts - Wifi
Pinouts - Various

Xboo Multiboot Cable
AUX Xboo PC-to-GBA Multiboot Cable
AUX Xboo Flashcard Upload
AUX Xboo Burst Boot Backdoor
DS Xboo

AUX GBA Game Pak Bus

Game Pak Bus - 32pin cartridge slot
The cartridge bus may be used for both CGB and GBA game paks. In GBA mode, it is used as follows:

```
Pin     Name     Dir  Expl.
1       VDD      O    Power Supply 3.3V DC
2       PHI      O    System Clock (selectable none, 4.19MHz, 8.38MHz, 16.78MHz)
3       /WR      O    Write Select    ;\latched address to be incremented on
4       /RD      O    Read Select     ;/rising edges of /RD or /WR signals
5       /CS      O    ROM Chip Select ;-A0..A15 to be latched on falling edge
6-21    AD0-15   I/O  lower 16bit Address    and/or  16bit ROM-data (see below)
22-29   A16-23   I/O  upper 8bit ROM-Address   or    8bit SRAM-data (see below)
30      /CS2     O    SRAM Chip Select
31      /REQ     I    Interrupt request (/IREQ) or DMA request (/DREQ)
32      GND      O    Ground 0V
```

When accessing game pak SRAM, a 16bit address is output through AD0-AD15, then 8bit of data are transferred through A16-A23.
When accessing game pak ROM, a 24bit address is output through AD0-AD15 and A16-A23, then 16bit of data are transferred through AD0-AD15.
The 24bit address is formed from the actual 25bit memory address (byte-steps), divided by two (halfword-steps).
Pin Pitch is 1.5mm.

8bit-Gamepak-Switch (GBA, GBA SP only) (not DS)
A small switch is located inside of the cartridge slot, the switch is pushed down when an 8bit cartridge is inserted, it is released when a GBA cartridge is inserted (or if no cartridge is inserted).
The switch mechanically controls whether VDD3 or VDD5 are output at VDD35; ie. in GBA mode 3V power supply/signals are used for the cartridge slot and link port, while in 8bit mode 5V are used.
The switch additionally drags IN35 to 3V when an 8bit cart is inserted, the current state of IN35 can be determined in GBA mode via Port 4000204h (WAITCNT), if the switch is pushed, then CGB mode can be activated via Port 4000000h (DISPCNT.3), this bit can be set ONLY by opcodes in BIOS region (eg. via CpuSet SWI function).
In 8bit mode, the cartridge bus works much like for GBA SRAM, however, the 8bit /CS signal is expected at Pin 5, while GBA SRAM /CS2 at Pin 30 is interpreted as /RESET signal by the 8bit MBC chip (if any). In practice, this appears to result in 00h being received as data when attempting to read-out 8bit cartridges from inside of GBA mode.

AUX DS Game Card Slot

```
Pin     Dir  Name  Connection in cartridge
1   >   -    GND   (ROM all unused Pins, EEPROM Pin 4 = VSS)
2       Out  CLK   (4MB/s, ROM Pin 5, EEPROM Pin 6 = CLK)
3   N   -    ?     (ROM Pin 17) (Seems to be not connected in console)
4   i   Out  /CS1  (ROM Pin 44) ROM Chipselect
5   n   Out  /RES  (ROM Pin 42) Reset, switches ROM to unencrypted mode
6   t   Out  /CS2  (EEPROM Pin 1) EEPROM Chipselect
7   e   In   IRQ   (GND)
8   n   -    3.3V  (ROM Pins 2, 23, EEPROM Pins 3,7,8 = /W,/HOLD,VCC)
9   d   I/O  D0    (ROM Pin 18)
10  o   I/O  D1    (ROM Pin 19)
```

```
11     I/O  D2    (ROM Pin 20)
12  C  I/O  D3    (ROM Pin 21)
13  0  I/O  D4    (ROM Pin 24)
14  1  I/O  D5    (ROM Pin 25)
15  -  I/O  D6    (ROM Pin 26, EEPROM Pin 2 = Q = Data EEPROM to NDS)
16  0  I/O  D7    (ROM Pin 27, EEPROM Pin 5 = D = Data NDS to EEPROM)
17  1  -    GND   (ROM all unused Pins, EEPROM Pin 4 = VSS)
```

Chipselect High-to-Low transitions are invoking commands, which are transmitted through data lines during next following eight CLK pulses, after the command transmission, further CLK pulses are used to transfer data, the data transfer ends at chipselect Low-to-High transition.
Data should be stable during CLK=LOW period throughout CLK rising edge.
Note: Supply Pins (1,8,17) are slightly longer than other pins. Pin pitch is 1.5mm.

The DS does also have a 32pin cartridge slot, that slot is used to run GBA carts in GBA mode, it can be also used as expansion port in DS mode.

AUX Link Port

Serial Link Port Pin-Out (GBA:"EXT" - GBA SP:"EXT.1")

```
Pin  Name  Cable
1    VDD35 N/A       GBA Socket     GBA Plug   Old "8bit" Plug
2    SO    Red       ___________    _________   ___________
3    SI    Orange   |  2  4  6  |  / 2  4  6 \  |  2  4  6  |
4    SD    Brown     \_1_ 3 _5_/   \_1_ 3 _5_/   \_1__3__5_/
5    SC    Green        '-'           '-'
6    GND   Blue      Socket Outside View / Plug Inside View
Shield     Shield
```

Note: The pin numbers and names are printed on the GBA mainboard, colors as used in Nintendo's AGB-005 and older 8bit cables.

Serial Link/Power Supply Port (GBA-Micro: "EXT.")

```
1  In  DC (Supply 5.2VDC)  ____________________
2  Out V3 (SIO 3.3VDC)    |  1 2 3 4 5 6 7 8  |
3  I/O SO (SIO RCNT.3)    | ================= |
4  I/O SI (SIO RCNT.2)     \__________________/
5  I/O SD (SIO RCNT.1)
6  I/O SC (SIO RCNT.0)
7  OUT DG (SIO GROUND)
8  In  DG (Supply GROUND)
-  -   -  (Shield not connected)
```

Cable Diagrams (Left: GBA Cable, Right: 8bit Gameboy Cable)

```
Big Plug  Middle Socket  Small Plug    Plug 1         Plug 2
 SI ____________________     ____ SI     SI ______  ______SI
 SO _____________SO   |__ | ___ SO     SO ______><______SO
 GND_____________GND______|____GND     GND_____________GND
 SD _____________SD_____________ SD     SD               SD
 SC _____________SC_____________ SC     SC _____________ SC
 Shield_______Shield_______Shield     Shield_______Shield
```

Normal Connection
Just connect the plugs to the two GBAs and leave the Middle Socket disconnected, in this mode both GBAs may behave as master or slave, regardless of whether using big or small plugs.
The GBA is (NOT ???) able to communicate in Normal mode with MultiPlay cables which do not have crossed SI/SO lines.

Multi-Play Connection

Connect two GBAs as normal, for each further GBAs connect an additional cable to the Middle socket of the first (or further) cable(s), up to four GBAs may be connected by using up to three cables.
The GBA which is connected to a Small Plug is master, the slaves are all connected to Large Plugs. (Only small plugs fit into the Middle Socket, so it's not possible to mess up something here).

Multi-Boot Connection
MultiBoot (SingleGamepak) is typically using Multi-Play communication, in this case it is important that the Small plug is connected to the master/sender (ie. to the GBA that contains the cartridge).

Non-GBA Mode Connection
First of all, it is not possible to link between 32bit GBA games and 8bit games, parts because of different cable protocol, and parts because of different signal voltages.
However, when a 8bit cartridge is inserted (the GBA is switched into 8bit compatibility mode) it may be connected to other 8bit games (monochrome gameboys, CGBs, or to other GBAs which are in 8bit mode also, but not to GBAs in 32bit mode).
When using 8bit link mode, an 8bit link cable must be used. The GBA link cables won't work, see below modification though.

Using a GBA 32bit cable for 8bit communication
Open the middle socket, and disconnect Small Plugs SI from GND, and connect SI to Large Plugs SO instead. You may also want to install a switch that allows to switch between SO and GND, the GND signal should be required for MultiPlay communication only though.
Also, cut off the plastic ledge from the plugs so that they fit into 8bit gameboy sockets.

Using a GBA 8bit cable for 32bit communication
The cable should theoretically work as is, as the grounded SI would be required for MultiPlay communication only. However, software that uses SD for Slave-Ready detection won't work unless when adding a SD-to-SD connection (the 8bit plugs probably do not even contain SD pins though).

AUX Sound/Headphone Socket and Battery/Power Supply

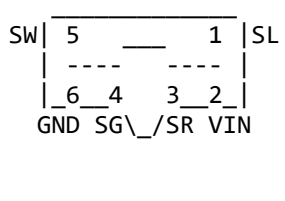
GBA, GBA-Micro, NDS, and NDS-Lite: Stereo Sound Connector (3.5mm, female)

Tip	Audio Left	
Middle	Audio Right	
Base	Ground	

The NDS socket doesn't fully match regular 3.5mm plugs, one needs to cut-off a portion of the DS case to be able to fully insert the plug, which still requires a lot of pressure, furthermore, when fully inserted, left/right become shortcut to mono, so one needs to pull-back the plug a bit to gain stereo output.

GBA SP and NDS - Power/Headphone Socket (EXT.2)

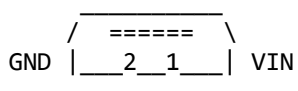
Pin	SP	NDS	Expl.
1	P31	SL	Audio LOUT
2	P32	VIN	Supply Input (DC 5.2V)
3	P33	SR	Audio ROUT
4	P34	SG	Audio GND (via 100uF to GND)
5	P35	SW	Audio Speaker Disable (GND=Dis)
6		GND	Supply GND
Shield			GND



External power input is used to charge the built-in battery, it cannot be used to run the SP without that battery.

NDS-Lite - Power Socket

Pin	Expl.
1	Supply Input (DC 5.2V)
2	Supply GND



GBA-Micro - Power Socket
Uses an 8pin socket (which combines SIO and Power), for pin-outs, see
AUX Link Port

External Power Supply
GBA: DC 3.3V (no separate power socket, requires 2xAA-battery-shaped adapter)
GBA-SP/NDS: DC 5.2V (or DC 5V) (special connector on power/headphone socket)
NDS-Lite: DC 5.2V (or DC 5V) (another special connector on power socket)

Internal Battery Supply
GBA: 2xAA (3V)
GBA-SP: Li-ion 3.7V, 600mAh (built-in, recharge-able)
GBA-Micro: Li-ion 3.8V, 460mAh (built-in, recharge-able)
NDS: Li-ion 3.7V, 850mAh (built-in, recharge-able)
NDS-Lite: Li-ion 3.7V, 1000mAh (built-in, recharge-able)

Using PC +5V DC as Power Supply
Developers whom are using a PC for GBA programming will probably want to use the PC power supply (gained from disk drive power supply cable) for the GBA as well rather than dealing with batteries or external power supplies.
GBA: To lower the voltage to approximately 3 Volts use two diodes, type 1N 4004 or similar, the ring printed onto the diodes points towards the GBA side, connected as such:

```
PC +5V (red)   --------|>|---|>|--------  GBA BT+
PC GND (black) ------------------------  GBA BT-
```

GBA SP, GBA Micro, NDS, and NDS-Lite: Works directly at +5V connected to EXT.2 socket (not to the internal battery pins), without any diodes.

AUX DSi SD/MMC Pin-Outs

SD/MMC Transfer Modes

Transfer Modes	SPI-Mode	1-bit-Bus	4-bit-Bus	SDIO
MMC Cards	Optional	Yes	MMCplus	No
SD Cards	Yes	Yes	Optional??	Optional

Note: SDIO is an extension to the SD protocol, allowing to access other non-memory-card hardware (such like cameras or network adaptors) via SD connectors.
Note: Original MMC cards don't support 4-bit bus, but there are revisions like MMCplus and MMCmobile (with extra pin rows) which do support 4-bit and 8bit bus.

SD/MMC Pin-Outs

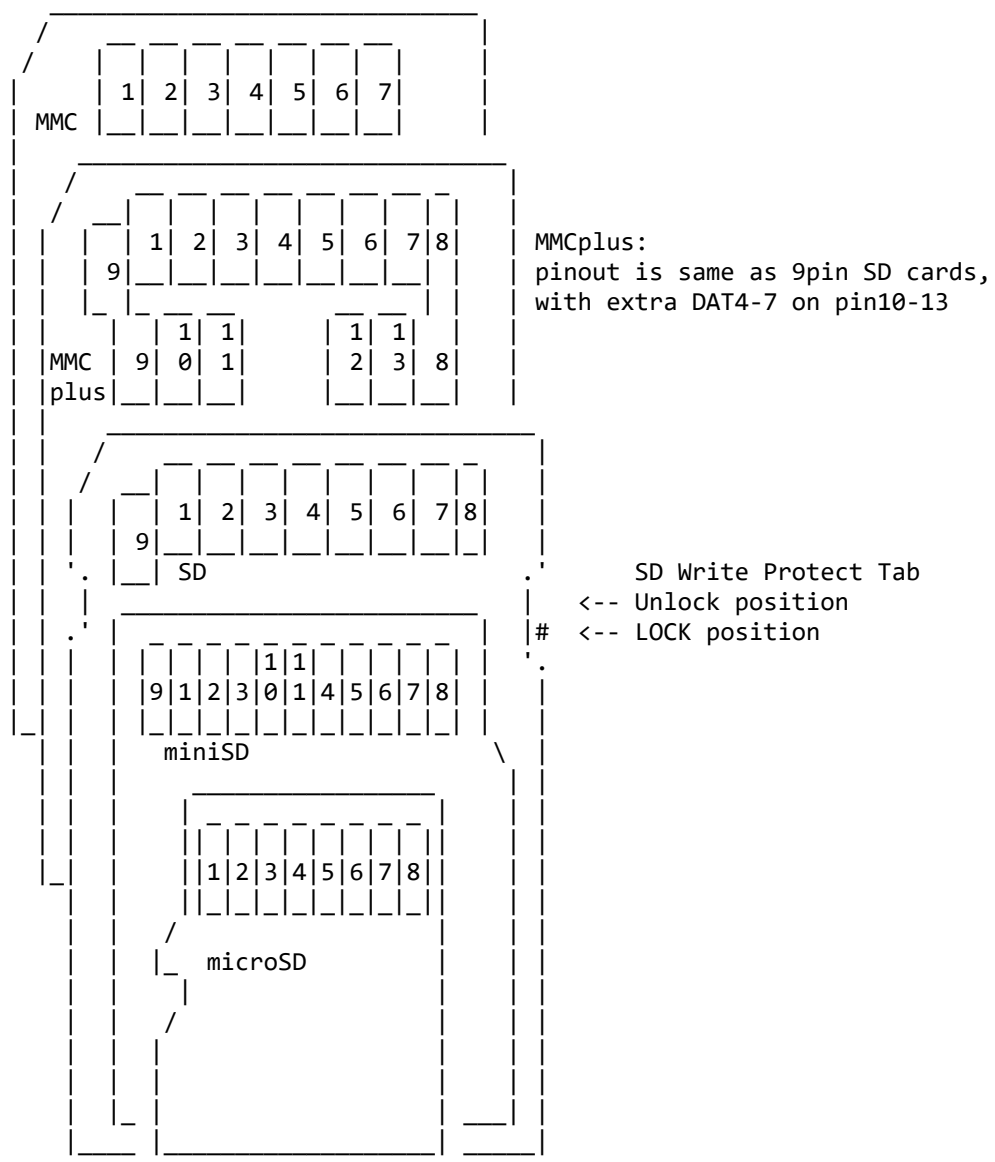
MMC	MMCplus	SD	miniSD	microSD	SPI-Mode	1-bit-Bus	4-bit/8bit-Bus
1	1	1	1	2	/CS	CardDetect	Data3
2	2	2	2	3	DataIn	CMD/REPLY	CMD/REPLY
3	3	3	3	--	GND	GND	GND
4	4	4	4	4	VDD	VDD	VDD
5	5	5	5	5	CLK	CLK	CLK
6	6	6	6	6	GND	GND	GND
7	7	7	7	7	DataOut	Data	Data0
--	8	8	8	8	/IRQ (SDIO)	/IRQ (SDIO)	Data1 or /IRQ (SDIO)
--	9	9	9	1	NC	NC	Data2
--	10	--	--	--	NC	NC	Data4 ;\
--	11	--	--	--	NC	NC	Data5 ; MMCplus
--	12	--	--	--	NC	NC	Data6 ; 8bit
--	13	--	--	--	NC	NC	Data7 ;/
--	--	--	10	--	Reserved	Reserved	Reserved
--	--	--	11	--	Reserved	Reserved	Reserved

Moreover, the card sockets (not the cards themselves) are usually containing a Card Detect switch, and, for SD card sockets, also a write protect switch:

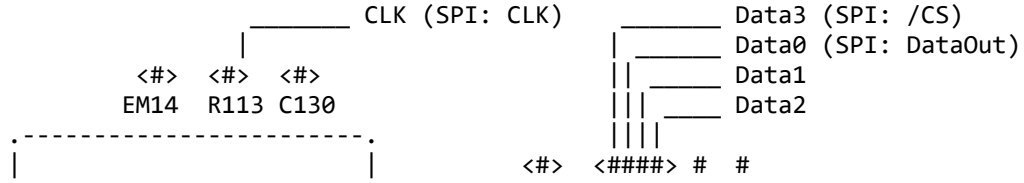
```
--   --      CD  CD     CD      Card Detect (senses if card is inserted)
--   ---     WP  --     --      Write Protect (senses position of LOCK tab)
```

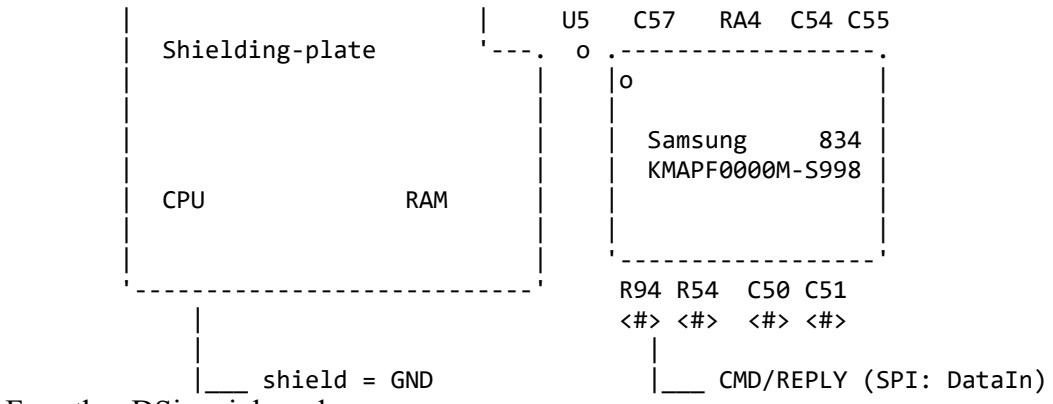
Note that the LOCK tab on SD cards is just a small piece of plastic without any electronics attached to it, the actual switch/sensor is located in the SD card socket (ie. the LOCK works much like the write-protect tabs on audio tapes, video tapes, and floppy discs).
Card detect can be an actual switch (however, some sockets are simply having dual contacts for Pin 3 (GND), one being GND, and the other becoming GNDed when a cartridge is inserted).

SD/MMC Card Shapes



SD/MMC Signals for on-board eMMC chip on DSi Mainboard "C/TWL-CPU-01"
Below are the required eMMC signals. Low-end hardware may get away with using Data0 as single data line (eg. small microprocessors with few I/O pins), but higher quality hardware should support 4bit data mode (eg. off-the-shelve SD card interfaces may insist on all four data lines being connected). Data3 (aka CardDetect) might be also needed even in 1bit data mode.





For other DSi mainboards, see:

http://www.dsibrew.org/wiki/Hardware

The KMAPF0000M chip does probably NOT support SPI mode, and it does probably support only MMC protocol (not SD protocol). That, assuming that the chip does have similar capabilities as in KMCEN0000M datasheet (there's no KMAPF0000M datasheet online).

SD/MMC Signals and 3DS Hardmodding

Pictures with eMMC solder pads for 3DS mainboards boards can be found at:

https://www.3dbrew.org/wiki/Hardware

However, hardmodding is much more difficult for 3DS as than for DSi. Three 3DS-specific problems:

CLK resistor

The 3DS tends to hold CLK low, and there is no resistor installed between 3DS CPU and eMMC chip, so the card reader may be unable to pull CLK high. As workaround, install a resistor in the CLK line (the mainboard has square solder pads just for that purpose) (330..660 ohms have worked for me, I am currently using 470 ohms; smaller resistors like 0..120 ohm won't work with the card reader, and much larger resistors won't work with the 3DS itself).

Incompatible Card readers

Newer 3DS/New3DS models seem to contain eMMC chips that are incompatible with older card readers, maybe because their firmware/driver treats the CSD version number as unknown, despite of the eMMC chip being nothing special. As workaround, try to use some newer card readers, or use a DSi console as card reader (ie. write you software that can happily ignore CSD version).

Mechanical problems

Most solder points are on the wrong PCB side (unlike as on DSi, you will need to remove the 3DS mainboard). Whereas, the connector for the top screen backlight is rated to survive max 10 mating cycles (mine died after 5-7). Part of the problem was that I wasn't aware of the other problems mentioned above (so I had removed the mainboard several times to check my soldering), and, the connector is 'underneath' of the PCB, making it difficult to insert the cable straight and smoothly, causing pins in the connector may get bent, making it impossible to insert the ribbon cable (maybe it will help if you take time, and don't impatiently try to push the cable into the socket).

Soldering Notes

Connect CLK/CMD to the pins on right side of R113/R94 (as shown in the drawing). Connect Data3/0/1/2 to the LOWER pins of RA4 (unlike as shown in the drawing, ie. NOT to the upper pins), or alternately, connect them to the four vias below of RA4. Connect GND somewhere to shielding plate, for example.

My own setup is: A 8pin ribbon cable soldered to a spare SD-to-SDmicro adapter (used as connector for SD/MMC slots), the ribbon cable is wired to a small circuit board, which is soldered to the shielding of the DSi's game cartridge slot (just for mechanical stability). Next, some fragile wires are forwarded from the circuit board to the actual mainboard pins.

Software Notes

Remove the DSi wifiboard (not absolutely required, but doing so will hang the DSi before accessing the eMMC, which ensures that the eMMC won't be accessed simultaneously by the DSi and PC). Switch on the DSi. Connect it to SD/MMC card reader. Under Windows, the eMMC should show up as MMC-storagedevice in

Windows Explorer (alongsides with your HDD drives), due to the encryption it isn't possible to access the filesystem or logical partitions of the chip. However, the physical sectors can be accessed.
For example, using HxD hex editor: Click Extras, Open Disk, and select the MMC (in HxD it shows up under Physical Discs: as Removeable Disk). Click Edit, Select All, Copy. Click File, New. Then Edit, Paste. And finally File, Save As for saving an image of the whole 240MByte FLASH chip.
I've tried accessing the eMMC on two PCs, one worked, the other didn't:

```
Win98 with External Card reader: Windows didn't recognize the MMC chip
Win7  with External Card reader: Okay (recognized as "unformatted" disk)
Win7  with Internal Card reader: Okay (recognized as "unformatted" disk)
```

For testing the Operating System/Card Reader side: Connect a normal SD card to the card reader. If HxD is showing it as both Logical Disc and Physical Disc, then you are fine. If it shows up as Logical Disc only, then your setup won't work for accessing the eMMC chip.

AUX Opening the GBA

Since Nintendo uses special screws with Y-shaped heads to seal the GBA (as well as older 8bit gameboys), it's always a bit difficult to loosen these screws.

Using Screwdrivers
One possible method is to use a small flat screwdriver, which might work, even though it'll most likely damage the screwdriver.
Reportedly, special Y-shaped screwdrivers for gameboys are available for sale somewhere (probably not at your local dealer, but you might find some in the internet or elsewhere).

Destroying the Screws
A more violent method is to take an electric drill, and drill-off the screw heads, this might also slightly damage the GBA plastic chase, also take care that the metal spoons from the destroyed screws don't produce shortcuts on the GBA mainboard.

Using a selfmade Screwdriver
A possible method is to take a larger screw (with a normal I-shaped, or X-shaped head), and to cut the screw-tip into Y-shape, you'll then end up with an "adapter" which can be placed in the middle between a normal screwdriver and gameboy screws.
Preferably, first cut the screw-tip into a shape like a "sharp three sided pyramid", next cut notches into each side.
Access to a grinding-machine will be a great benefit, but you might get it working by using a normal metal-file as well.

Opening the GBA Micro
- open the case with appropriate screwdriver or drilling machine or whatever
- remove the plastic front-plate (there are two snap-ins inside at ONE side)
- remove the mainboard and screen and plastic skeleton from the metal case
- remove the start/select daughter-board from the plastic skeleton
- remove the plastic skeleton (move the screen through the skeleton)
- remove the screen (lift lcd socket front-side, backlight socket rear-side)

Opening the NDS-Lite
- open the case with appropriate screwdriver or drilling machine or whatever
- remove the RFU unit, and the 4-pin touch-screen cable (under the RFU unit)
- remove the mainboard together with the lower screen
- remove the upper/lower screen cables (on the rear-side of the mainboard)

AUX Mainboard

Other possibly useful signals on the mainboard...

FIQ Signal
The FIQ (Fast Interrupt) signal (labeled FIQ on the mainboard) could be used as external interrupt (or debugging break) signal.
Caution: By default, the FIQ input is directly shortcut to VDD35 (+3V or +5V power supply voltage), this can be healed by scratching off the CL1 connection located close to the FIQ pin (FIQ still appears to have an internal pull-up, so that an external resistor is not required).
The GBA BIOS rejects FIQs if using normal ROM cartridge headers (or when no cartridge is inserted). When using a FIQ-compatible ROM header, Fast Interrupts can be then requested by pulling FIQ to ground, either by a push button, or by remote controlled signals.

RESET Signal
The RESET signal (found on the mainboard) could be used to reset the GBA by pulling the signal to ground for a few microseconds (or longer). The signal can be directly used (it is not shortcut to VDD35, unlike FIQ).
Note: A reset always launches Nintendo's time-consuming and annoying boot/logo procedure, so that it'd be recommend to avoid this "feature" when possible.

Joypad Signals
The 10 direction/button signals are each directly shortcut to ground when pressed, and pulled up high otherwise (unlike 8bit gameboys which used a 2x4 keyboard matrix), it'd be thus easy to connect a remote keyboard, keypad, joypad, or read-only 12bit parallel port.

AUX DSi Component Lists

DSi Mainboard "C/TWL-CPU-01" Components

```
U1 352pin   CPU TWL       (under shielding plate)                    ;\under
U2   ?pin   RAM 8Mx16, Fujitsu MB82DBS08164D-70L, NEC uPD46128512AF1 ;/shield
U3  56pin   "TexasIns 72071B0" or "Mitsumi 3317A" (powerman?) (right of NAND)
U4  48pin   "AIC3000D, TI 89K, EXDK G4" (PAIC3 codec? above headphone socket)
U5   ?pin   Samsung KMAPF0000M-S998 (eMMC, 256Mbyte NAND FLASH)
U6  36pin   "BPTWL, K007K, 0902KM00D" (small/square, left of cartridge slot)
U7   8pin   "AOK, S8BXS" (ISL95810, I2C potentiometer)         ;\on PCB
U8   8pin   "7BDS" (PCA9306, I2C voltage translator)           ;/backside
U9  12pin   "199A, 01IU" (Seiko S-35199A01) (RTC) ;under shielding plate (A)
U10  4pin   "6800" or "688F" Hinge Magnet Sensor (PCB backside, near A/B/X/Y)
U11 10pin   ",\\ 908, 335A" or "2005D, 8350" (right of cartridge slot)
U12  5pin   "L8NX" or "C7JHN" (upper-right of PCB back-side) ;text layer (B)
U13  5pin   Backlight 1, "U01" or "KER"  ;\lower-right board edge
U14  5pin   Backlight 2, "U01" or "KER"  ;/see text-layer (B)
U15  4pin   ",\\ T34" (near external power input)
U16     -   N/A
U17  6pin   "VY" or "Z198" (in lower-right, on PCB backside)
U18  6pin   "YJ" (above headphone socket)
U19  5pin   "E30H6" or "L2SX" (at lower right of cartridge slot)
Q1   6pin   external power supply related
Q2    pin   N/A ?
Q3   6pin   ... above battery plug
Q4   3pin   maybe MUTE for SR  ;\old TWL-CPU-01 mainboard only
Q5   3pin   maybe MUTE for SL  ;/(replaced by Q17?/Q18? on newer boards)
Q6   6pin   MC1_VDD power ON (supply)
Q7   3pin   MC1_VDD power OFF (pulldown)
Q8    pin   N/A ?
Q9    pin   N/A ?
Q10   pin   N/A ?
```

```
  Q11  3pin   BLUE (LED)   ;\LEDs (note: the other LEDs, ORANGE
  Q12  3pin   YELLOW (LED) ;      and YELLOW, are driven directly)
  Q13  3pin   CAM_LED      ;/
  Q14  3pin   not installed (above powerman chip)
  Q15  3pin   not installed (above powerman chip)
  Q16  3pin   VDD-5 related, near DPAD socket
  Q17? 6pin   maybe MUTE  ;\  ;\new TWL-CPU-10 mainboard only
  Q18? 6pin   maybe MUTE  ;/  ;/(formerly Q4/Q5 on older boards)
  X1   4pin   16.756  (rectangular oscillator)        ;\under shielding plate
  X2   4pin   CB837 or CB822 (long slim osc) for RTC? ;/text layer: see (A)
  F1   2pin   Fuse for external power input
  SW1  2pin   Button A (right)
  SW2  2pin   Button B (lower)
  SW3  2pin   Button X (upper)
  SW4  2pin   Button Y (left)
  SW5  2pin   Button Select (lower)
  SW6  2pin   Button Start  (upper)
  P1  19pin   NDS/DSi cartridge slot (17pin slot + 2pin switch at right side)
  P2      -   N/A
  P3      -   N/A
  P4   8pin   External microphone/headphone combo socket
  P5  50pin   Wifi-Daughterboard
  P6      -   N/A
  P7  47pin   To UPPER lcd screen (video+backlight+speakers) (on PCB backside)
  P8  37pin   To LOWER lcd screen (video signals)
  P9  25pin   To UPPER lcd screen (signals for both cameras, and camera led)
  P10  4pin   To LOWER lcd screen (touchpad X-,Y-,X+,Y+)
  P11  2pin   External Power Supply input (4.6V DC IN)
  P12     -   N/A
  P13     -   N/A
  P14     -   N/A
  P15 15pin   To battery/DPAD/PowerButton board (and onwards to 3xLEDs)
  P16 26pin   To bottom cover (SD Slot and L/R/VOL+/- buttons)
  P17  2pin   Battery cable (lower-right) ;see text-layer (B)
  P18  4pin   To LOWER lcd screen (backlight cathode/anode)
  P19  1pin   Shielding-Plate for CPU (lower clip)
  P20  1pin   Shielding-Plate for CPU (upper clip)
  P21  1pin   Shielding-Plate for CPU (right clip)
  P22     -   N/A
  P23  2pin   To Internal Microphone (via orange shielded wire)
```

DSi Front/bottom-Side Text Layer sections (in upper left of mainboard)

```
  (A)  For components underneath of shielding plate
  (B)  For components in lower-right board edge (near battery connector)
  (C)  For components at middle/right of cartridge slot
  (D)  For components left of U4 (left board edge)
  (E)  For components right of U4 (above headphone socket)
  (F)  For components at lower/right of cartridge slot
```

DSi Back/top-Side Text Layer sections (at various places)

```
  (A)  at top/middle,  for components at upper right edge
  (B)  at middle/left, for components near upper right edge
  (C)  at lower/left,  for components left of Y-button
  (D)  at lower/righz, for components at right edge
```

DSi Wifi Daughterboard (DWM-W015) (older DSi version)

PCB Text: "RU (S)-717V 01 ,\\" or "RU (S)-717V 03 ,\\" or "FK RU 06 ,\\"

```
  U   56pin "Mitsumi, Japan, 844L, MM3218" (same as in DS Lite)
  U  132pin "ROCm, Atheros, AR6002G-AC1B, E19077.1B, 0844, Taiwan"
  U    8pin I2C EEPROM "408F, B837" (HN58X2408F; 1Kx8 for atheros calibration)
  U    8pin SPI FLASH big chip "45PE10V6, HPASC VS, KOR 8364, ST"  ;\either one
  U    8pin SPI FLASH tiny chip "5A32, 8936?"                      ;/installed
  U    8pin "4P, K" or "S6, K" (odd 3+1+3+1 pin package, near antenna)
  U    4pin "3VP, OT" or "3VB, OS" (at board edge, near 50pin connector)
  X    4pin "26.000, 9848" (bigger oscillator, for atheros chip)
  X    4pin "22.000, xxxx" (smaller oscillator, for mitsumi chip)
```

```
  P   50pin Connector to Mainboard
  P    2pin Connector for Antenna (shielded white cable)
```
The "3VP/3VP" thing is some 1.2V voltage regulator (sth like LP3983 or TPS799xx or similar).
White PCB sticker (underneath of the black isolation sticker): "DWM-W015, IC:4250A-DWMW015, FCC ID:EW4DWMW015, [R]003WWA080444, [T]D080261003, MADE IN PHILIPINES, MITSUMI ELEC. CO., LTD."

DSi Wifi Daughterboard (DWM-W024) (newer DSi version)
PCB Text: "FB RU 06 ,\\"
```
  U   76pin "ROCm, Atheros, AR6013G-AL1C" (or 80pin, with 4pins at edges?)
  U    8pin I2C EEPROM? "4DA?, D940?"   ;maybe I2C eeprom for atheros
  U    8pin SPI FLASH "5A32, 8937?"   ;FLASH (small solder pads)
  U    8pin SPI FLASH not installed (alternate bigger solder pads for FLASH?)
  U    4pin "?" (at board edge, near 50pin connector)
  X    4pin "??" (oscillator, near ROCm chip)
  P   50pin Connector to Mainboard
  P    2pin Connector for Antenna (shielded white cable)
```

DSi Wifi Daughterboard (J27H020) (alternate newer DSi version)
This seems to be functionally same as DWM-W024, but the PCB is different, and the circuit seems to use a different voltage regulator, and includes solder pads for some additional uninstalled components. And the shielding is difficult to remove because it's soldered all way round (rather than just at some solder points).
```
 Sticker 1:
  HON HAI PRECISION IND.CO.,LTD.        ;aka Foxconn
  MODEL: J27H020
  [R] 003WWA100195
  [T] D100196003
  FCC ID: MCLJ27H020
  IC: 2878D-27H020
 Sticker 2:
  (barcode)
  <mac.address> J27H020.00 LF, 4xxxxxx-xxx
 Board:
  U1   76pin atheros ... "ATHEROS, AR6013G-AL1C, N2U586.00C, 1035, KOREA"
  U2    -    -
  U3    -    -
  U4    8pin I2C EEPROM "G80, 8, G02"
  U5    8pin SPI FLASH big chip "26FV032T, OGK01"  (installed)
  U6    -    -
  U7    8pin SPI FLASH small chip (not installed)
  U?    5pin voltage regulator or so "IG19P"
  U?    6pin whatever near antenna (not installed)
  Y1    4pin crystal "H400K"
  J3    2pin Connector for Antenna (shielded white cable)
  J6   50pin Connector to Mainboard
```

3DS Wifi Daughterboard (DWM-W028)
Component list is unknown. The thing is said to use a "Atheros AR6014" chip.
Later 3DS models have the Wifi unit (with AR6014G chip) on the mainboard (instead of using a removeable DWM board).

DSi Battery/DPAD Daughterboard "C/TWL-SUB-01"
```
  TH1  2pin  Battery Thermal Sensor maybe? (about 10kOhm at room temperature)
  F1   2pin  Battery Fuse
  SW1  2pin  DPAD Up Button
  SW2  2pin  DPAD Down Button
  SW3  2pin  DPAD Left Button
  SW4  2pin  DPAD Right Button
  SW5  2pin  Power/Reset Button
  P1  15pin  To Mainboard (P15) (button/led signals) (wire "15P-01")
  P2   6pin  To 3xLEDs
```

```
P3   3pin  To battery (TWL-003 3.6V 840mAh 3Wh C/TWL-A-BP, Li-ion 00"
Wire 2pin  To Mainboard (P17) (battery supply) (red=vcc, black=gnd)
```

DSi LED Board/Foil "LED-01, (DF)"

```
D    2pin  Left LED      ;-wifi
D    2pin  Middle LED    ;-charge
D    2pin  Right LED 1   ;\power "two-color-LED"
D    2pin  Right LED 2   ;/composed of 2 single LEDs
Wire 6pin  To Battery/DPAD Daughterboard
```

DSi Lower Screen with Touchpad

```
Wire 4pin  Touchpad
Wire 4pin  Backlight (actually 2pins, each 2 pins are same)
Wire xxpin Video Signals
LCD  "LS033A1DG48R, 8X16Q U0003986"
```

DSi Upper Screen with Speakers & Cameras & LED & Microphone

```
Orange Ribbon Cable: Video Signals, Backlight, and Speakers
Black Ribbon Cable: Cameras and Camera LED
Shielded Orange 2pin Wire: Microphone
Shielded White 2pin Wire: Wifi PCB Antenna
LCD  "LS033A1DG38R, BX16Q L0005532"
The speakers use red/black wires, which connect to the orange ribbon cable
```

DSi Upper Screen Area Extra Components: Speakers & Cameras & LED & Microphone

DSi Lower Case (SD Slot, L/R and VOL+/- Buttons, and screen calibration)

```
Whatever, not checked yet
```

DSi Disassembly Notes

Bottom Cover Screws:

```
7 screws (two are under battery cover)
```

Remove bottom cover, and:

```
P16: To bottom cover (SD Slot and L/R/VOL+/-) --> pull (away from board)
```

Remove Wifi Daughterboard:

```
P5:  Wifi-board (without cable)              --> pull (away from board)
WHITE: Wifi-Antenna (shielded 2pin)          --> pull (away from wifi-board)
```

Remove mainboard:

```
ORANGE P24 (shielded 2pin)  --> pull (away from board)
WHITE SUPPLY                --> lift (use screwdriver & push away from board)
3x bigger white/black connectors --> lift black lid (at cable-side)
2x smaller black connectors      --> lift black lid (at cable-end) (!!!)
```

Turn mainboard over, then unlock the connector at back side:

```
1x bigger white/black connector  --> lift black lid (at cable-side)
```

Remove Battery board:

```
1x smaller black connector       --> lift black lid (at cable-end) (!!!)
1x bigger white/black connectors --> lift black lid (at cable-side)
(don't disconnect bigger connector if the other cable end is already
disconnected from mainboard) (or if you did do, reassemble as follows:
longer cable end to battery board, short cable end to mainboard)
1x battery cable (disconnect at mainboard side, see there)
```

Top Cover Disassembly:

```
Disconnect upper LCD and mic/antenna from mainboard (see above)
Remove 4 screws (all hidden under square rubber pieces)
slide rear bezel upwards by two millimeters?
push metal hinge inwards by three millimeters (under LED unit)
```

AUX DSi Internal Connectors

P1 - 19pin - NDS/DSi cartridge slot (17pin slot + 2pin switch at right side)

```
1   GND
2   MC1_CLK
3   -
4   MC1_CS
5   MC1_RES
6   MC1_CS2
7   MC1_IREQ
8   MC1_VDD via Q6 to VDD33 (cpu signal preamplified from Q7)
9   MC1_IO0
10  MC1_IO1
11  MC1_IO2
12  MC1_IO3
13  MC1_IO4
14  MC1_IO5
15  MC1_IO6
16  MC1_IO7
17  GND
18  MC1_DET           ;\switch closed when cart inserted
19  GND               ;/
Shield GND
```

P4 - 8pin - External microphone/headphone combo socket

```
1  GND    ;\      ;\
2  SL     ; head- ; headphone gnd/left/right
3  SR     ; phone ;/
4  GND    ;       ;\headphone/speaker switch (pin 4+5 shortcut with each
5  HP#SP  ;/      ;/other when no headphone connected)
6  MIC    ;\    ;\microphone switch (pin6+7 shortcut when no mic connected)
7  Switch ; mic ;/(internal mic from P23 is then passed from pin7 to pin6)
8  GND    ;/
```

P5 - 50pin - DSi Wifi-Daughterboard (DWM-W015, DWM-W024, or J27H020)

```
                          GND   2  1   SDIO.CLK       ;\
                        VDD18   4  3   GND            ; SDIO for
                        VDD18   6  5   SDIO.DAT0      ; Atheros Wifi
                          GND   8  7   SDIO.DAT3      ; (CLK, CMD, DATA0-3)
                        VDD33  10  9   SDIO.DAT1      ;
                        VDD33  12  11  SDIO.CMD       ;
                          GND  14  13  SDIO.DAT2      ;/
                     ATH_TX_H  16  15  DSi: NC (DWM: JTAG_TDO)
                    /WIFI_RST  18  17  DSi: NC (DWM: JTAG_TMS)
    (DWM:JTAG_TDI)    DSi: NC  20  19  GND
    (DWM:JTAG_TCK)    DSi: NC  22  21  RTC_FOUT (or RTC_F32K?)  ;for Atheros?
    (DWM:JTAG_TRST_L) DSi: NC  24  23  GND
 (near CPU irq pins) SEL_ATH_L  26  25  DSi: NC (wifi: via 0 ohm MM3218.pin47)
              /FLASH_WP (R122) 28  27  SPI_CS2 (wifi FLASH memory)
                       SPI_SCK  30  29  BBP_SLEEP    to MM3218.pin42
                      SPI_MISO  32  31  RF_SLEEP     to MM3218.pin41
                      SPI_MOSI  34  33  RF_SCS       to MM3218.pin38
      to MM3218.pin15     CCA  36  35  BBP_SCS      to MM3218.pin37
                      WL_RXPE  38  37  BB_RF_SDO    to MM3218.pin36
      to MM3218.pin19  TRDATA  40  39  BB_RF_SDI    to MM3218.pin35
                          GND  42  41  BB_RF_SCLK   to MM3218.pin34
      to MM3218.pin21   TRCLK  44  43  NC(VDD18_TP) to MM3218.pin28 (0ohm+cap)
      to MM3218.pin18   TRRDY  46  45  GND
                      WL_TXPE  48  47  MCLK    to MM3218.pin23 (via XX & CLxx)
                        RESET  50  49  GND
```

https://fccid.io/EW4DWMW024/Label/Label-format-and-location-1137926.pdf
https://fccid.io/EW4DWMW015/Label/Label-Location-1031953.pdf
Pin 25 and 43 are VDD test points on DWM-W015 wifiboard (but are NC on DSi mainboard & DWM-W024 wifiboard).

P7 - 47pin - To UPPER lcd screen (video+backlight+speakers) (on PCB backside)

```
 BLA2   1  2   BLC2    ;-backlight
 SPLN   3  4   SPLN    ;\left speaker
 SPLP   5  6   SPLP    ;/
 SPRN   7  8   SPRN    ;\right speaker
 SPRP   9  10  SPRP    ;/
VDD-5  11  12  VDD10
 VDD5  13  14  GND
 VSHD  15  16  VSHD
  INI  17  18  GSP
  GCK  19  20  LDB20
LDB21  21  22  LDB22
LDB23  23  24  LDB24
LDB25  25  26  LDG20
LDG21  27  28  LDG22
  GND  29  30  LDG23
LDG24  31  32  LDG25
LDR20  33  34  LDR21
LDR22  35  36  LDR23
LDR24  37  38  LDR25
  GND  39  40  DCLK
  SPL  41  42  LS
  GND  43  44  via C79 to COM2
  REV  45  46  GND
 COM2  47
```

P8 - 37pin - To LOWER lcd screen (video signals)

```
VDD-5   1  2   VDD10
 VDD5   3  4   GND
 VSHD   5  6   VSHD
  INI   7  8   GSP
  GCK   9  10  LDB10
LDB11  11  12  LDB12
LDB13  13  14  LDB14
LDB15  15  16  LDG10
LDG11  17  18  LDG12
  GND  19  20  LDG13
LDG14  21  22  LDG15
LDR10  23  24  LDR11
LDR12  25  26  LDR13
LDR14  27  28  LDR15
  GND  29  30  DCLK
  SPL  31  32  LS
  GND  33  34  via C93 to COM1
  REV  35  36  GND
 COM1  37
```

P9 - 25pin - To UPPER lcd screen (signals for both cameras, and camera led)

```
          GND   1  2   CAM_LED
        VDD42   3  4   GND
R100     RCLK   5  6   GND
          GND   7  8   HSYNC
        VSYNC   9  10  CAM_D5  RA7
 RA7  CAM_D6   11  12  CAM_D4  RA7
     CAM_RST   13  14  SCL
          SDA  15  16  CAM_D7  RA7
 RA6  CAM_D0   17  18  CAM_D3  RA6
 RA6  CAM_D1   19  20  CAM_D2  RA6
        VDD28  21  22  GND
          CKI  23  24  GND
        VDD18  25
```

P10 - 4pin - To LOWER lcd screen (touchpad X-,Y-,X+,Y+)

```
1 X-
2 Y-
3 X+
4 Y+
```

P11 2pin External Power Supply input (4.6V DC IN)

```
1 VIN  (+4.6V)
2 VGND (GND)
Shield (GND)
```

P15 - 15pin - To battery/DPAD/PowerButton board (and onwards to 3xLEDs)

```
dpad up button      P06  1  2  ORANGE (LED) Battery Charge
dpad right button   P04  3  4  BLUE   (LED) Power On/Good
dpad left button    P05  5  6  YELLOW (LED) Wifi
dpad down button    P07  7  8  RED    (LED) Power On/Low
                    GND  9  10 VDD42 (to LEDs)
                    GND 11  12 TH on DPAD board (via R102 to TH on main)
middle battery pin  DET 13  14 GND
power button       PWSW 15
```

Note: On Daughterboard, pins are mirrored (eg. PWSW=Pin1 instead Pin15)

P16 - 26pin - To bottom cover (SD Slot and L/R/VOL+/- buttons)

```
                    GND  2  1  SD10_CLK                   ;\
             SD10_DATA0  4  3  SD10_VDD (aka VDD33)       ;
             SD10_DATA1  6  5  SD10_VDD (aka VDD33)       ; pin 1..18
                SD10_WP  8  7  GND                        ; to RIGHT side:
                    GND 10  9  SD10_CMD                   ; R-button, and
shoulder button R   P08 12  11 GND                        ; SD-card slot
                    GND 14  13 SD10_DATA3                 ;
                SD10_CD 16  15 SD10_DATA2                 ;
                    GND 18  17 GND                        ;/
                    GND 20  19 GND                        ;\pin 19..20
maybe display ;\   VDD5 22  21 VOLP (aka volume plus?)    ; to LEFT side:
calibration?  ;    COM1 24  23 VOLN (aka volume minus?)   ; L-button, VOL +/-
(at battery)  ;/   COM2 26  25 P09   shoulder button L    ;/and calibration
```

P17 - 2pin - Battery cable (lower-right) ;see text-layer (B)

```
+ BT+ (plus) (red wire)
- BT- (GND)  (black wire)
```

P18 - 4pin - To LOWER lcd screen (backlight cathode/anode)

```
1 BLC1  ;\both same
2 BLC1  ;/
3 BLA1  ;\both same
4 BLA1  ;/
```

P19 - 1pin - Shielding-Plate for CPU (lower clip)
P20 - 1pin - Shielding-Plate for CPU (upper clip)
P21 - 1pin - Shielding-Plate for CPU (right clip)

```
Shield GND
```

P23 - 2pin - To Internal Microphone (via orange shielded wire)

```
Pin  MIC (from P4.Pin7, disconnected when external microphone connected)
Shield GND
```

DPAD-BOARD - P2 - 6pin - To LEDs

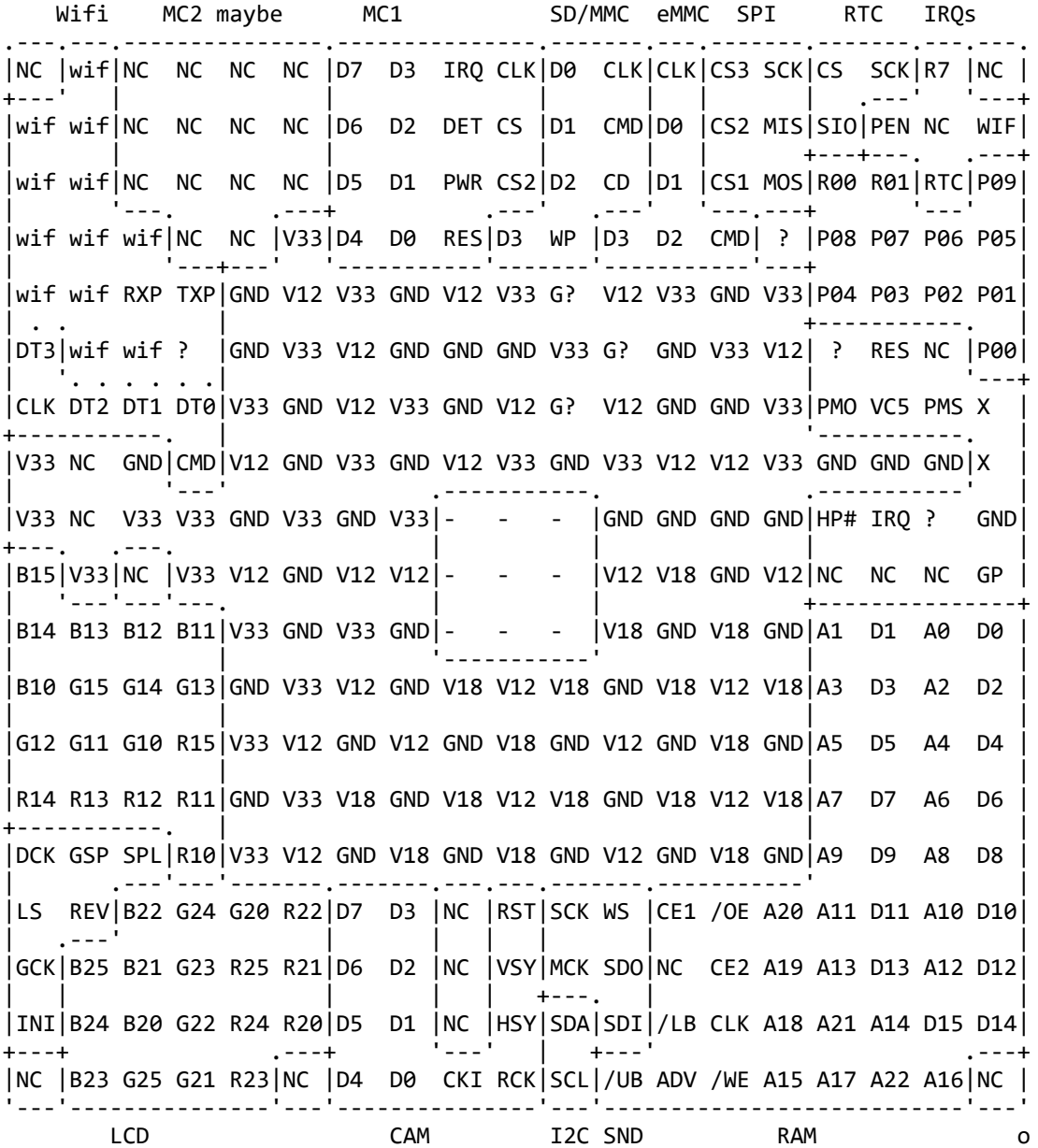
```
1  YELLOW  Wifi
2  BLUE    Power On/Good
3  ORANGE  Battery Charge
```

```
4  GND   (for red+orange)
5  RED     Power On/Low
6  VDD42 (for yellow+blue)
```

AUX DSi Chipset Pinouts

A photo of the DSi mainboard (with extra text layer on vias and solderpads) can be found at:
`http://problemkaputt.de/twl-core.jpg`

DSi U1 - TWL-CPU (19x19 pin grid) (352 pins, aka 19x19 minus middle 3x3 pins)



DSi U2 - Main RAM (8Mx16) (Fujitsu MB82DBS08164, or NEC uPD46128512)

DSi mainboard solder pads (12x8 grid, within 14x10 grid):

	A	B	C	D	E	F	G	H	J	K	L	M	N	P
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	NC	NC	-	A15	A21	A22	A16	NC	VSS	-	NC	NC	-
8	-	NC	NC	A11	A12	A13	A14	NC	DQ15	DQ7	DQ14	NC	NC	-
7	-	-	-	A8	A19	A9	A10	DQ6	DQ13	DQ12	DQ5	-	-	-
6	-	-	-	/WE	CE2	A20	-	-	DQ4	VDD	NC	-	-	-
5	-	-	-	CLK	/ADV (W)	-	-	DQ3	VDD	DQ11	-	-	-	

```
  4    -    -    -    /LB  /UB  A18  A17  DQ1  DQ9  DQ10 DQ2  -    -    -
  3    -    NC   -    A7   A6   A5   A4   VSS  /OE  DQ0  DQ8  NC   NC   -
  2    -    NC   NC   -    A3   A2   A1   A0   NC   /CE1 -    NC   NC   -
  1 o  -    -    -    -    -    -    -    -    -    -    -    -    -    -
```

DSi RAM - Fujitsu MB82DBS08164 (14x10 grid):

```
       A    B    C    D    E    F    G    H    J    K    L    M    N    P
  10   NC   NC   NC   NC   NC   NC   VDD  VSS  NC   NC   NC   NC   NC   NC
  9    NC   NC   NC   NC   A15  A21  A22  A16  NC   VSS  NC   NC   NC   NC
  8    -    -    NC   A11  A12  A13  A14  NC   DQ15 DQ7  DQ14 NC   -    -
  7    -    -    NC   A8   A19  A9   A10  DQ6  DQ13 DQ12 DQ5  NC   -    -
  6    -    -    NC   /WE  CE2  A20  NC   NC   DQ4  VDD  NC   NC   -    -
  5    -    -    NC   CLK  /ADV /WAI NC   VDD  DQ3  VDD  DQ11 VDD  -    -
  4    -    -    NC   /LB  /UB  A18  A17  DQ1  DQ9  DQ10 DQ2  VSS  -    -
  3    -    -    VSS  A7   A6   A5   A4   VSS  /OE  DQ0  DQ8  NC   -    -
  2    NC   NC   NC   NC   A3   A2   A1   A0   NC   /CE1 NC   NC   NC   NC
  1 o  NC   NC   -    NC   NC   NC   VDD  VSS  NC   NC   NC   NC   NC   NC
```

DSi RAM - NEC uPD46128512 (14x10 grid):

```
       A    B    C    D    E    F    G    H    J    K    L    M    N    P
  10   NC   NC   NC   -    -    -    NC   NC   -    -    -    NC   NC   NC
  9    -    NC   NC   -    A15  A21  A22  A16  NC   VSS  -    NC   NC   -
  8    -    -    NC   A11  A12  A13  A14  NC   DQ15 DQ7  DQ14 NC   -    -
  7    -    -    -    A8   A19  A9   A10  DQ6  DQ13 DQ12 DQ5  -    -    -
  6    -    -    NC   /WE  CE2  A20  NC   NC   DQ4  VCC  NC   NC   -    -
  5    -    -    NC   CLK  /ADV /WAI NC   NC   DQ3  VCC  DQ11 NC   -    -
  4    -    -    -    /LB  /UB  A18  A17  DQ1  DQ9  DQ10 DQ2  -    -    -
  3    -    -    NC   A7   A6   A5   A4   GND  /OE  DQ0  DQ8  NC   -    -
  2    -    NC   NC   NC   A3   A2   A1   A0   NC   /CE1 -    NC   NC   -
  1 o  NC   NC   NC   -    -    -    NC   NC   -    -    -    NC   NC   NC
```

The Fujitsu & NEC datasheets are specifing 14x10 grid (the chips are essentially pin-compatible, except that: NEC has removed some NC pins, changed some supply pins to NC, and (attempted to) rename VSS to GND). However, the DSi mainboard has 12x8 grid solder pads (and the installed NEC/Fujitsu chip/packages are actually only 12x8, too).

The DSi debug version is said to have 32Mbyte RAM, there is no provision for that feature on DSi retail boards; dev boards are probably somewhere having an extra address line (or an extra chip select for a 2nd RAM chip).

DSi U3 - Power Managment (Texas Instruments 72071B0, or Mitsumi 3317A)

```
  1   GND (via CL9)
  2   ADPO
  3   EXTB+
  4   VDD33
  5   RESET          ;\
  6   SPI_SCK        ; main cpu bus
  7   SPI_MOSI       ; (reset and spi)
  8   SPI_MISO       ;
  9   SPI_CS1        ;/  <-- powerman (this does ALSO connect to U4)
  10  GND
  11  PMOFF
  12  PWSWO
  13  VCNT5
  14  PM_SLP
  ---
  15  B+
  16  VDD12 via L1
  17  VDD12
  18  GND
  19  BLC1           ;\
  20  BLA1 via U13   ; backlight 1+2
  21  BLA2 via U14   ; anode/cathode
  22  BLC2           ;/
  23  GND
  24  B+
  25  B+
  26  VDD18 via L2
```

```
27  VDD18 via L2
28  VDD18
---
29  DET     ;\battery contacts
30  BT+     ;/    ;\these are almost shortcut
31  VDET-         ;/with each other (via 0 ohm R71)
32  PVDD
33  PWSW (when off: very few ohms to PVDD)
34        ... via R104 (100K) to Q3 (B+ enable or so?)
35  B+
36        ... via to C18 to GND (seems to have no other connection)
37  GND
38  AOUT  ;\to U6
39  GND   ;
40  SCL1  ;   ;\secondary IC2 bus (to U6)
41  SDA1  ;/  ;/
42  VDD33
---
43  GND via CL10
44  VDD5 input (sense if VDD5/C16 has reached voltage)
45  charge-pump for VDD5 (L7 and via DA3 to VDD5/C16)
46  charge-pump for VDD5 (L5 and C14)
47  VDD33 (via CL5)
48  VDD33 (via L3)
49  VDD33
50  VDD33
51  B+
52  B+
53  B+
54  charge-pump for VDD42 (L7 and C23)
55  charge-pump for VDD42 (L7 and via D3 to VDD42/C22)
56  VDD42 input (sense if VDD42/C22 has reached voltage)
```

DSi U4 - Sound and Touchscreen controller (AIC3000D)

AIC3000D pinout is same as in TSC2117 datasheet (aside from GPIx/GPIOx pins).

```
Pin TSC2117 AIC3000D
1   MISO    SPI_MISO
2   MOSI    SPI_MOSI
3   /SS     SPI_CS1 (powerman, this does ALSO connect to U3)
4   SCLK    SPI_SCLK
5   GPIO1   SPI_CS3 (touchscreen)
6   GPIO2   PENIRQ
7   IOVSS   GND
8   IOVDD   VDD33
9   DVDD    VDD18
10  SDOUT   SND_SDI             ;\
11  SDIN    SND_SDO             ;
12  WLCK    SND_WS              ; serial sound input from main cpu
---                             ; (and serial output? microphone maybe?)
13  BCLK    SND_SCLK            ;
14  MCLK    SND_MCLK            ;/
15  SDA       ... via R107 to VDD18  ;\unused I2C bus (?)
16  SCL       ... via R106 to VDD18  ;/
17  VOL/M   wiper (sound volume, from I2C potentiometer) "VOL/MICDET?"
18  MICBIAS   LIN-related-1 ... to 6pin U18
19  MIC     LIN (aka MIC via C31)
20  AUX1      LIN-related-2   ;\via 0ohm R108 to ... something on U18
21  AUX2      LIN-related-2   ;/                    that is almost GND
22  AVSS    GND
23  AVDD    VDD33
24  VBAT    GND
---
25  VREF    VDD33
26  TSVSS   GND
```

```
27  YN      Y-                  ;\
28  XN      X-                  ;
29  DVSS    GND                 ; touchscreen input
30  YP      Y+                  ;
31  XP      X+                  ;/
32  TSVDD   VDD33
33  SPLN    SPLN                ;\
34  SLVSS   GND                 ;
35  SLVDD   B+                  ; speaker output
36  SPLP    SPLP                ;
---                             ;
37  SPRN    SPRN                ;
38  SRVDD   B+                  ;
39  SRVSS   GND                 ;
40  SPRP    SPRP                ;/
41  HPL     SL via CP2 and R88      ;\
42  HVDD    VDD33                   ;
43  HVSS    GND                     ; headphone output
44  HPR     SR via CP3 and R89      ;
45  GPI3    MUTE via Q4/Q5 to SR/SL ;
46  GPI2    HP#SP switch            ;/
47  GPI1    VCNT5
48  /RESET  RESET
```

DSi U5 - 256Mbyte eMMC NAND (14x14 grid)

```
      1    2    3    4    5    6    7    8    9    10   11   12   13   14
o A   NC   NC   DAT0 DAT1 DAT2 NC   NC   NC   NC   NC   NC   NC   NC   NC
  B   NC   DAT3 DAT4 DAT5 DAT6 DAT7 NC   NC   NC   NC   NC   NC   NC   NC
  C   NC   VDDI NC   VSSQ NC   VCCQ NC   NC   NC   NC   NC   NC   NC   NC
  D   NC   NC   NC   NC   -    -    -    -    -    -    -    NC   NC   NC
  E   NC   NC   NC   -    NC   VCC  VSS  NC   NC   NC   -    NC   NC   NC
  F   NC   NC   NC   -    VCC  -    -    -    -    NC   -    NC   NC   NC
  G   NC   NC   NC   -    VSS  -    -    -    -    NC   -    NC   NC   NC
  H   NC   NC   NC   -    NC   -    -    -    -    VSS  -    NC   NC   NC
  J   NC   NC   NC   -    NC   -    -    -    -    VCC  -    NC   NC   NC
  K   NC   NC   NC   -    NC   NC   NC   VSS  VCC  NC   -    NC   NC   NC
  L   NC   NC   NC   -    -    -    -    -    -    -    -    NC   NC   NC
  M   NC   NC   NC   VCCQ CMD  CLK  NC   NC   NC   NC   NC   NC   NC   NC
  N   NC   VSSQ NC   VCCQ VSSQ NC   NC   NC   NC   NC   NC   NC   NC   NC
  P   NC   NC   VCCQ VSSQ VCCQ VSSQ NC   NC   NC   NC   NC   NC   NC   NC
```

Note: The pinout follows JEDEC's eMMC standard. The "NC" pins are GNDed in DSi, the "DAT4..DAT7" pins are not connected in DSi. The "VDDI" pin isn't wired to VDD, instead it goes to a capacitor (0.1uF min) "for internal power stability".

DSi U6 - "BPTWL" - I2C bus(ses), LEDs, volume, power, wifi (6x6 grid)

```
      GND          WL_TXPE      P02(button) SDA'33       ADPO         GND        o
      ATH_TX_H     BLUE(LED)    RED(LED)    SCL'33       V33          GND
      YELLOW(LED)  VOLP button  VOLN button  PM_SLP       V33'         /WIFI_RST
      SDA1         RESET        SCL1        to C46       GND          to U17
      VDD28        GND          CAM_LED     PWSWO        mFE          /mRST
      GND          AOUT         mFE'(R79)   WL_RXPE      /IRQ_O       GND
```

DSi U7 - I2C bus potentiometer (ISL95810) (Device 50h)

```
1  /WP (DSi: VDD33)  writeprotect
2  SCL (DSi: SCL1)   i2c bus   ;\from U6
3  SDA (DSi: SDA1)   i2c bus   ;/
4  GND (DSi: GND)    ground
5  RW  (DSi: wiper)  pot.wiper ;-to U4
6  RL  (DSi: VDD18)  pot.L
7  RH  (DSi: GND)    pot.H
8  VCC (DSi: VDD33)  supply
```

DSi U8 - bidirectional I2C voltage translator (PCA9306)

```
1  GND   (DSi: GND)
2  VREF1 (DSi: VDD18)
3  SCL1  (DSi: SCL)     ;\to U1 (CPU)
4  SDA1  (DSi: SDA)     ;/
5  SDA2  (DSi: SDA'33)  ;\to U6 (LED/stuff)
6  SCL2  (DSi: SCL'33)  ;/
7  VREF2 (DSi: VDD33)
8  EN    (DSi: VDD33)
```

DSi U9 - RTC - Seiko S-35199A01 (4x3 grid)

```
      A    B    C    D
3     CS   /SCK VDD  F32K
2     SIO  CTRL /INT FOUT
1 o   VSS  XIN  XOUT VDDL
```

DSi U10 - Magnet Sensor (for hinge, aka shell opened/closed)

```
1  VDD33
2  R7 (HINGE)  ;to U1
3  GND
4  GND
```

DSi U11 - charge

```
1  EXTB+
2  Rosc
3  ORANGE (via R2)              ;-charge LED
4  GND
5  TH' (via R76 to TH)          ;\thermal sensor
6  TH  (via R102 to DPAD board) ;/for battery?
7  B+  (?)
8  RICHG
9  BT+
10 BT+
```

Note: Seems to resemble Mitsumi MM3358 datasheet.

DSi U12 - 5pin, VDD33 to VDD28 converter? (near upper screen socket)
DSi U13 - 5pin, Backlight 1, near power managment chip
DSi U14 - 5pin, Backlight 2, near power managment chip
DSi U15 - 4pin, something near external power input
DSi U16 - N/A
DSi U17 - 6pin, something near headphone socket, connects to U6, and MUTE
DSi U18 - 6pin, something near headphone socket, MIC/LIN related
DSi U19 - 5pin, something near dpad socket

Smaller misc chips.

DSi LEDs:

```
CAM_LED (via R68 and Q13)  ;\
BLUE    (via R21 and Q11)  ; from U6
YELLOW  (via R22 and Q12)  ;/
RED     (via R20)          ;-from U6 (or to U6 ?)
ORANGE  (via R2)           ;-from U11
```

DSi Wifi Daughterboard - MM3218 chip (same chip as in DS Lite)

```
1  VDD18
2  GND
3  VDD18
4  Antenna signal
5  Antenna shield
6  VDD18
7  VDD18
8  GND
9  NC
```

```
 10 GND
 11 GND
 12 GND
 13 NC
 14 /RESET
 ---
 15 CCA    ... to DSi mainboard connector pin 36
 16 WL_TXPE
 17 WL_RXPE
 18 TRRDY  ... to DSi mainboard connector pin 46 !!!
 19 TRDATA ... to DSi mainboard connector pin 40
 20 VDD33
 21 TRCLK  ... to DSi mainboard connector pin 44
 22 GND
 23 MCLK   ... via nearby big component ... to DSi mainboard connector pin 47
 24 VDD18
 25 NC
 26 22MHz
 27 22MHz'
 28 ... to DSi mainboard connector pin 43 NC? (with cap to GND and via 0 ohm)
 ---
 29 VDD33
 30 via capacitor to VDD33
 31 via 1K2 + 120K to GND (aka via 121.2K to GND)
 32 VDD18
 33 VDD18
 34 BB_RF_SCLK   ... to DSi mainboard connector pin 41
 35 BB_RF_SDI    ... to DSi mainboard connector pin 39
 36 BB_RF_SDO    ... to DSi mainboard connector pin 37
 37 BBP_SCS      ... to DSi mainboard connector pin 35
 38 RF_SCS       ... to DSi mainboard connector pin 33
 39 GND
 40 VDD33
 41 RF_SLEEP     ... to DSi mainboard connector pin 31
 42 BBP_SLEEP    ... to DSi mainboard connector pin 29
 ---
 43 VDD18
 44 ... shortcut to MM3218.pin50, and via resistor to MM3218.pin46
 45 VDD33
 46 ... via resistor to MM3218.pin44+50
 47 ... to DSi mainboard connector pin 25 (via 0 ohm) (+cap) (NC in DSi)
 48 VDD33
 49 GND
 50 ... shortcut to MM3218.pin44, and via resistor to MM3218.pin46
 51 ... via resistor to GND
 52 VDD18
 53 NC
 54 NC
 55 NC
 56 NC
```

DSi DWM-W015 Wifi Daughterboard - ROCm Atheros AR6002G-AC1B chip

	1	2	3	4	5	6	7	8	9	10	11	12	13
A	AGND	RF2 OUTN	RF2 OUTP	RF2 INP	RF2 INN	PDET	NC	NC	VDD18 BIAS	VDD12 XTAL	XTAL I	XTAL O	BT_CLK OUT
B	RF5 INP	AGND	VDD18 FE	VDD12 LNA	VDD12 BIAS	BIAS REF	NC	NC	VDD12 D_SYN	VDD12 BB	VDD18 XTAL	BT_ CLKEN	DVDD12
C	RF5 INN	VDD12 FE	-	-	AGND	AGND	AGND	AGND	AGND	-	-	GPIO 17	GPIO 16
D	PA5 BIAS	NC	-	-	-	-	-	-	-	-	-	GPIO 14	GPIO 15
E	RF5 OUT	VDD18 VCO	AGND	-	AGND	AGND	AGND	AGND	DVSS	-	-	DVDD GPIO1	DVDD GPIO0
F	VDD12	VDD12	AGND	-	AGND	AGND	AGND	AGND	DVSS	-	DVSS	GPIO	GPIO

```
   TX5   SYNTH                                                          12    13
G XPA   XPA   AGND  -     AGND  AGND  AGND AGND DVSS  -     DVSS  GPIO  GPIIO
  BIAS2 BIAS5                                                         10    11
H VCCFE LDO_  AGND  -     AGND  AGND  AGND AGND DVSS  -     DVSS  GPIO  DVDD
  M     OUT                                                           9     12
J ANTA  VDDIO AGND  -     DVSS  DVSS  DVSS DVSS DVSS  -     DVSS  CLK_  DVDD
        ANT                                                           REQ   12
K ANTC  ANTB  -     -     -     -     -    -    -     -     -     SYS_  CHIP_
                                                                    RST_L PWD_L
L ANTD  ANTE  -     -     DVSS  DVSS  DVSS  DVSS DVSS -     -     DVDD  DVDD_
                                                                    12    SDIO
M AGND  GPIO0 GPIO2 DVDD  GPIO4 GPIO6 GPIO8 JTAG JTAG JTAG  SDIO_ SDIO_ SDIO_
                    12                      TMS  TCK  TDO   DATA3 DATA2 CLK
N DVDD  GPIO1 GPIO3 DVDD_ GPIO5 GPIO7 DVDD_ DVDD JTAG DVDD_ SDIO_ SDIO_ SDIO_
  12                BT                SDIO  12   TDI  SDIO  CMD   DATA1 DATA0
```

DSi DWM-W024 Wifi Daughterboard - Atheros AR6013 chip

```
 1  1.2V
 2  VDD18
 3  NC
 4  NC
 5  VDD18
 6  NC
 7  1.2V
 8  VDD18
 9  1.2V
 10 NC
 11 NC (except, wired to tespoint)
 12 VDD33
 13 via 0 ohm to ATH_TX_H  ;<--with 0 ohm connection
 14 via (N/A) to ATH_TX_H  ;<--connection not installed
 15 to a dead-end-via
 16 to a dead-end-via
 17 1.2V
 18 P5.pin24       JTAG_TRST_L     (plus testpoint)
 19 NC (except, wired to tespoint)
 ---
 20 VDD18
 21 1.2V
 22 I2C.SCL
 23 I2C.SDA
 24 P5.pin21 RTC 32KHZ via 0 ohm
 25 /WIFI_RST
 26 ATH_TX_H
 27 SDKI.CMD
 28 SDIO.CLK
 29 VDD33
 30 SDIO.DAT0
 31 SDIO.DAT2
 32 SDIO.DAT1
 33 SDIO.DAT3
 34 P5.pin22       JTAG_TCK
 35 P5.pin20       JTAG_TDI
 36 P5.pin17       JTAG_TMS
 37 1.2V
 38 P5.pin15       JTAG_TDO
 ---
 39 P5.pin36       CCA
 40 VDD33
 41 1.2V
 42 P5.pin46       TRRDY
 43 P5.pin44       TRCLK
 44 P5.pin40       TRDATA
 45 WL_TXPE
```

```
 46 WL_RXPE
 47 P5.pin47       MCLK
 48 P5.pin33       RF_SCS      ... and 6.9ohm to P5.47 ?
 49 P5.pin35       BBP_SCS
 50 P5.pin39       BB_RF_SDI
 51 VDD33
 52 P5.pin37       BB_RF_SDO
 53 P5.pin41       BB_RF_SCLK
 54 P5.pin29       BBP_SLEEP
 55 P5.pin31       RF_SLEEP
 56 P5.pin26       SEL_ATH_L   ... IRQ?
 57 1.2V
 ---
 58 VDD18
 59 XTALx
 60 XTALx
 61 1.2V
 62 1.2V
 63 VDD18
 64 NC (except, wired to tespoint)
 65 NC (except, wired to tespoint)
 66 NC (except, wired to tespoint)
 67 NC (except, wired to tespoint)
 68 NC
 69 via 6.1K to GND
 70 1.2V
 71 NC
 72 NC
 73 VDD18
 74 RF2.OUTx
 75 RF2.OUTx
 76 VDD18
 ---
 GND center plates
```

Note: /RESET connects to Wifi FLASH only, ie. NOT to the MM3218 clone within AR6013G (that's unlike as real MM3218).

3DS DWM-W028 Wifi Daughterboard - Atheros AR6014 chip
Pinouts unknown.

There is a 3rd part number, J27H020, made by hon hai (Foxconn) instead of Mitsumi.

Pinouts - CPU - Signal Summary

Advance Gameboy CPU Signal Summary
Cart Bus: D0-D7, A0-A15, /CS, /RD, /WR (different usage in GBA/DMG mode)
WRAM Bus: WA0-WA16, WD0-WD15, /WLB, /WUB, /WWE, /WOE (used in GBA mode only)
LCD Bus : LDR1-5, LDG1-5, LDB1-5, DCK, LP, PS, SPL, CLS, SPS, MOD, REVC
Joypad: TP0-3 (Buttons), TP4-7 (Directions), TP8-9 (L/R-Buttons, via R43/R44)
Serial Link: SC, SD (aka P14?), SI, SO - Audio: SO1-2, Vin
Other: CK1-2, PHI, IN35, VCNT5, /FIQ (via CL1 to VDD3), /RESET (IN), /RES (OUT)
Supply: VDD35, VDD3, VDD2, GND (some are probably undoc inputs)
GBA SP: Same as GBA, plus VDD1, plus duplicated supply pins, plus pin 152.

Pinouts - CPU - Pinouts

Advance Gameboy CPU Pinouts (CPU AGB)

```
  1 VDD3  17 D0   33 A0    49 WA4   65 VDD2  81 WD9   97 LDB5   113 CK1
  2 IN35  18 A15  34 /CS   50 WA5   66 WD5   82 WD1   98 LDB4   114 CK2
  3 TP8   19 A14  35 /RD   51 WA6   67 WD13  83 /WOE  99 LDB3   115 VDD2
  4 TP0   20 A13  36 /WR   52 WA7   68 WD6   84 DCK   100 LDB2  116 GND
  5 TP1   21 A12  37 PHI   53 /WLB  69 WD14  85 LP    101 LDB1  117 VDD2
  6 SO1   22 A11  38 VDD35 54 /WUB  70 WD7   86 PS    102 GND   118 VCNT5
  7 SO2   23 A10  39 GND   55 /WWE  71 WD15  87 LDR5  103 VDD3  119 TP9
  8 Vin   24 A9   40 SC    56 WA8   72 WD8   88 LDR4  104 SPL   120 TP6
  9 /RES  25 A8   41 SD    57 WA9   73 WD16  89 LDR3  105 CLS   121 TP5
  10 D7   26 A7   42 SI    58 WA10  74 WA16  90 LDR2  106 SPS   122 TP7
  11 D6   27 A6   43 SO    59 WA11  75 WD12  91 LDR1  107 MOD   123 TP4
  12 D5   28 A5   44 VDD2  60 WA12  76 WD4   92 LDG5  108 REVC  124 /FIQ
  13 D4   29 A4   45 WA0   61 WA13  77 WD11  93 LDG4  109 GNDed 125 /RESET
  14 D3   30 A3   46 WA1   62 WA14  78 WD3   94 LDG3  110 GNDed 126 TP2
  15 D2   31 A2   47 WA2   63 WA15  79 WD10  95 LDG2  111 GNDed 127 TP3
  16 D1   32 A1   48 WA3   64 GND   80 WD2   96 LDG1  112 GNDed 128 GND
```

GBA SP CPU Pinouts (CPU AGB B)

```
  1 IN35   21 D0    41 A0    61 WA4   81 WD13  101 GND   121 LDB4  141 GND
  2 TP8    22 A15   42 /CS   62 WA5   82 WD6   102 VDD1  122 LDB3  142 VDD3
  3 TP0    23 A14   43 /RD   63 WA6   83 WD14  103 GND   123 LDB2  143 GND
  4 TP1    24 A13   44 /WR   64 WA7   84 WD7   104 VDD3  124 LDB1  144 VCNT5
  5 SO1    25 A12   45 PHI   65 /WLB  85 WD15  105 DCK   125 GND   145 TP9
  6 SO2    26 A11   46 VDD35 66 /WUB  86 WD8   106 LP    126 VDD3  146 TP6
  7 Vin    27 GND   47 GND   67 GND   87 WD16  107 PS    127 SPL   147 TP5
  8 VDD1   28 VDD35 48 SC    68 VDD2  88 WA16  108 LDR5  128 CLS   148 TP7
  9 GND    29 A10   49 SD    69 /WWE  89 VDD2  109 LDR4  129 SPS   149 TP4
  10 VDD35 30 A9    50 SI    70 WA8   90 GND   110 LDR3  130 MOD   150 /FIQ
  11 /RES  31 A8    51 SO    71 WA9   91 WD12  111 LDR2  131 REVC  151 /RESET
  12 D7    32 A7    52 VDD35 72 WA10  92 WD4   112 LDR1  132 GND   152 ?
  13 D6    33 A6    53 GND   73 WA11  93 WD11  113 LDG5  133 GND   153 TP3
  14 D5    34 A5    54 VDD1  74 WA12  94 WD3   114 LDG4  134 GND   154 TP2
  15 D4    35 A4    55 GND   75 WA13  95 WD10  115 LDG3  135 GND   155 VDD3
  16 D3    36 GND   56 VDD2  76 WA14  96 WD2   116 LDG2  136 VDD1  156 GND
  17 D2    37 VDD35 57 WA0   77 WA15  97 WD9   117 LDG1  137 GND
  18 GND   38 A3    58 WA1   78 GND   98 WD1   118 GND   138 CK1
  19 VDD35 39 A2    59 WA2   79 VDD2  99 /WOE  119 VDD3  139 CK2
  20 D1    40 A1    60 WA3   80 WD5   100 VDD2 120 LDB5  140 VDD2
```

Pin 152 seems to be not connected on the mainboard, maybe an undoc output.

GBA-Micro, NDS, NDS-Lite, and DSi CPU Pinouts

Unknown. The CPU Pins are hidden underneath of the CPU. And, in NDS and NDS-Lite, the CPU itself hides underneath of the DS Cartridge Slot. In the DSi it's hidden underneath of a shielding plate (which is itself underneath of the removeable wifi daughterboard).

Pinouts - Audio Amplifiers

Advance Gameboy Audio Amplifier (AMP AGB IR3R60N) (U6)

```
  1   2   3   4   5   6   7   8   9   10  11  12   13   14   15   16   17  18
  C38 FR1 FR2 FL1 FL2 GND RIN LIN C39 VOL SW  VDD5 LOUT VCC3 ROUT VCC3 SP  GND
```

SW=Headphone Switch (grounded when none connected).

GBA SP Audio Amplifier (uses AMB AGB IR3R60N, too) (U3)

Same connection as in GBA, except that pin14/16 connect to VR21 (instead VCC3), and pin1/9 connect to different capacitors.

NDS - National Semiconductor LM4880M Dual 250mW Audio Power Amplifier (U12)

```
  1-OUT A  2-IN A  3-BYPASS  4-GND  5-SHUTDOWN  6-IN B  7-OUT A  8-VDD.VQ5
```

NDS-Lite: No external amplifier (Mitsumi 3205B Powermanagment Device contains internal amplifier).

Pinouts - LCD Cables

Advance Gameboy Display Socket

```
1 ?     6 GND    11 LDR2   16 LDG2   21 LDB3   26 SPS     31 P2-VSS  36 V4
2 VSHD  7 VSHD   12 LDR1   17 LDG1   22 LDB2   27 ?       32 P2-VCC  37 V3
3 DCK   8 LDR5   13 LDG5   18 GND    23 LDB1   28 MOD     33 ?       38 V2
4 LP    9 LDR4   14 LDG4   19 LDB5   24 SPL    29 VCOM    34 VDD5    39 V1
5 PS    10 LDR3  15 LDG3   20 LDB4   25 CLS    30 P2-VEE  35 GND     40 V0
```

GBA SP Display Socket

```
1 VSHD 5 VSHD  9 LDR3   13 LDG4  17 GND   21 LDB2  25 SPS   29 P2VSS 33 U83
2 DCK  6 GND   10 LDR2  14 LDG3  18 LDB5  22 LDB1  26 MOD   30 COM   34 VDD5
3 LP   7 LDR5  11 LDR1  15 LDG2  19 LDB4  23 SPL   27 REVC  31 VDD5
4 PS   8 LDR4  12 LDG5  16 LDG1  20 LDB3  24 CLS   28 P2VDD 32 GND
```

GBA Micro Display Sockets

```
__GBA Mirco display socket (P1)_____________________________________
1-PS    6-5bit  11-MD    16-5bit  21-5bit  26-CL             31-GND
2-RV    7-5bit  12-SL    17-5bit  22-5bit  27-SS             32-GND
3-GND   8-5bit  13-CK    18-5bit  23-5bit  28-via C5 to VR1  33-V10
4-5bit  9-LP    14-GND   19-5bit  24-5bit  29-V5             34-V-5
5-5bit  10-VD   15-5bit  20-GND   25-5bit  30-to VR1
__GBA Mirco backlight socket (P3)___________________________________
1-LC  2-LC  3-LA  4-LA
```

NDS Upper/Lower Display Sockets

```
___NDS upper screen/upper backlight/speakers socket (P3)_____________________
1-SPLO 7-PS2   13-LDR2  19-GND   25-LDG2  31-LDB2  37-MOD2  43-VDD15  49-SPRO
2-SPLO 8-REV2  14-LDR1  20-DCLK2 26-LDG1  32-LDB1  38-GND   44-VDD-5  50-GND
3-SSC2 9-GND   15-LDR0  21-GND   27-LDG0  33-LDB0  39-VDD5  45-VDD-10 51-GND
4-ASC2 10-LDR5 16-LS2   22-LDG5  28-LDB5  34-GCK2  40-VDD10 46-LEDC2
5-GND  11-LDR4 17-VSHD  23-LDG4  29-LDB4  35-GSP2  41-COM2  47-LEDA2
6-SPL2 12-LDR3 18-DISP1 24-LDG3  30-LDB3  36-GND   42-GND   48-SPRO
___NDS lower screen socket (P4)_____________________________________________
1-SSC1 6-REV1  11-LDR2  16-DISP0 21-LDG4  26-LDB5  31-LDB0  36-GND  41-VDD15
2-ASC1 7-GND   12-LDR1  17-SPL1  22-LDG3  27-LDB4  32-GCK1  37-?    42-VDD10
3-GND  8-LDR5  13-LDR0  18-DCLK1 23-LDG2  28-LDB3  33-GSP1  38-VDD5 43-GND
4-?    9-LDR4  14-LS1   19-GND   24-LDG1  29-LDB2  34-VSHD  39-COM1 44-VDD-5
5-PS1  10-LDR3 15-VSHD  20-LDG5  25-LDG0  30-LDB1  35-MOD1  40-GND  45-VDD-10
___NDS lower backlight socket (P5)____   ___NDS touchscreen socket (P6)______
1:LEDA1  2:LEDA1  3:LEDC1  4:LEDC1       1:Y-    2:X-    3:Y+    4:X+
```

NDS-Lite Upper/Lower Display Sockets

```
___NDS-Lite upper screen/upper backlight/speakers socket (P3)_________________
1-VDD-5 6-MOD    11-LD2xx 16-LD2xx 21-LD2xx 26-LD2xx 31-LS   36-GND   41-SPRO
2-VDD10 7-GSP    12-LD2xx 17-LD2xx 22-LD2xx 27-LD2xx 32-VSHD 37-COM2  42-SG
3-VDD5  8-GCK    13-LD2xx 18-GND   23-LD2xx 28-GND   33-GND  38-LEDA2 43-SG
4-GND   9-LD2xx  14-LD2xx 19-LD2xx 24-LD2xx 29-DCLK  34-xx2? 39-LEDC2 44-SPLO
5-VSHD  10-LD2xx 15-LD2xx 20-LD2xx 25-LD2xx 30-SPL   35-REV  40-SPRO  45-SPLO
___NDS-Lite lower screen/lower backlight (P4)_______________________________
1-VDD-5 6-MOD    11-LD1xx 16-LD1xx 21-LD1xx 26-LD1xx 31-LS   36-GND
2-VDD10 7-GSP    12-LD1xx 17-LD1xx 22-LD1xx 27-LD1xx 32-VSHD 37-COM1
3-VDD5  8-GCK    13-LD1xx 18-GND   23-LD1xx 28-GND   33-GND  38-LEDA1
4-GND   9-LD1xx  14-LD1xx 19-LD1xx 24-LD1xx 29-DCLK  34-xx1? 39-LEDC1
5-VSHD  10-LD1xx 15-LD1xx 20-LD1xx 25-LD1xx 30-SPL   35-REV
___NDS-Lite touchscreen socket (P6)______   ___NDS-Lite white coax (P12)_____
1:X-  2:Y-  3:X+  4:Y+                      Center:MICIN  Shield:GND
```

Pinouts - Power Switches, DC/DC Converters, Reset Generators

Advance Gameboy Power Switch (2-position slider, with two common pins)
GBA SP Power Switch (same as GBA)

```
  1 via resistor to GND (OFF)
  2 VS (BT+) (ON)
  C VCC (to board)
```

GBA Micro Power Switch
Same as GBA and GBA SP, but Pin 1 and 2 exchanged.

Advance Gameboy Cartridge Slot Switch (integrated 4pin micro switch)
GBA SP Cartridge Slot Switch (separate 4pin micro switch)

```
  C1 VDD35  (to S2 when PRESSED, to S1 when RELEASED)
  S1 VDD3   (to C2 when PRESSED, to C1 when RELEASED)
  C2 IN35   (to S1 when PRESSED)
  S2 VDD5   (to C1 when PRESSED)
```

Pressed=8bit DMG/MGB/CGB cart, Released=32bit GBA cart (or no cart inserted)
GBA: switch integrated in cart socket, GBA-SP: separate switch next to socket.

Advance Gameboy Power Controller (M 121 514X) (U4)

```
  1-VIN    2-VOUT5  3-CSS5    4-VDRV5  5-GND    6-VDRV3    7-CSS3    8-VOUT3
  9-VCNT5  10-CSCP  11-REGEXT 12-VDD3  13-VDD2  14-/RESET  15-LOWBAT 16-VDD13
```

/RESET is passed to the CPU, and then forwarded to /RES pin on cart slot.

Advance Gameboy LCD Regulator (AGB-REG IR3E09N) (U3)

```
  1  2  3    4      5   6   7   8   9   10   11  12  13  14  15  16   17     18
  ?  ?  REVC U3-COM V0  V1  ?   ?   ?   GND  ?   V2  ?   V3  V4  VDD5 U3-VDD ?
```

GBA SP Power Controller 1 (S6403 AU227 9276) (U4)

```
  1-VCC    2-SCP1   3-SCP2 4-VDRV3 5-VOUT3/VDD3 6-VDD2  7-VOUT1/VDD1  8-VDRV1
  9-LOWBAT 10-VCNT5 11-LS5 12-?    13-GND       14-?    15-VOUT5/VDD5 16-VDRV5
```

GBA SP Power Controller 2 (2253B 2808) (U5)

```
  1-TIN     2-U5C3     3-ADJ  4-U5VDD  5-VIN   6-?    7-U57  8-?
  9-to-C29  10-to-C30  11-?   12-GND   13-VS   14-S-  15-S+  16-U5OUT
```

GBA Micro - Power Managment Device (U2)

```
 1 via C43 to GND
 2 via R24 to C34 to R25 back to U2.2
 3 via C35 to GND
 4 via C36 to GND
 5
 6 audio.in ? (see BP)
 7   via C48 to GND
 8   via R21 to C46 to C47 to C38 to R23 to phones
 9  VL (to U4)
 10  via R27 to C33 to C44 to C49 to R22 to phones
 11  via C45 to GND
 12 audio.in ? (see BP)
 13  via C41 to GND
 14 phones (switch)
 15 phones (tip via R22)
 16 phones (mid via R23)
 17 VCS
 18 SP
 19 GND
```

```
20 LB
21 via C52 to GND
22 via C53 to GND
23 RS         (looks like RESET output)
24 to R37/C56 (looks like RESET input)
25
26
27 via C54 to V3
28 V3
29 GND
30 V3
31 VC
32 to C58
33 to R41/C58
34 GND
35
36 VC
37 VC
38
39 V5
40 GND
41 GND
42
43
44
45 B+
46 S-
47 S+
48
```

GBA Micro - Volume/Backlight Level Up/Down Controller (U5)

```
1-     5-GND  9-      13-XD      17-     21-     25-    29-
2-     6-GND  10-     14-to U4.7 18-XR   22-CN   26-    30-
3-     7-     11-XC   15-        19-V+   23-CNS  27-    31-BP
4-LN   8-     12-GND  16-        20-V-   24-     28-V3  32-
```

NDS Powermanagment Device (Mitsumi 3152A) (U3)

```
1 R50-EXTB+        17               33 LEDC1           49 VCNT5
2 R39-ORANGE       18               34 GND             50
3 GND              19 VQ5           35 LEDC2           51 RST
4                  20               36                 52
5 Rxx-Q4           21               37 U10-LEDA2       53
6 INS+             22 GND           38                 54
7 INS-             23 VQ5           39 MIC.C53-AIN     55 VQ5
8                  24               40 MIC.TSC.AUX     56 R24-SR
9 VDET             25 VDD3.3        41 GND             57
10 PVDD            26 GND           42 R38-RED         58 R22-SL
11                 27 CL60-VDD3.3   43 R37-GREEN       59 GND
12 PWSW            28 VSHD          44 VDD3.3          60 VR3.PIN2
13                 29               45 PWM.SPI.CLK     61
14 GND             30 VDD5          46 PWM.SPI.D       62
15 GND             31 U9-LEDA1      47 PWM.SPI.Q       63
16 VQ5             32               48 PWM.SPI.SEL     64 GND
```

NDS-LITE Powermanagment Device (Mitsumi 3205B) (U3)

```
1 SW               17               33 LEDC1           49 VCNT5
2 R50-EXTB+        18               34 GND             50
3 R39-ORANGE       19 VQ5           35 LEDC2           51 RST
4 GND              20               36                 52
5                  21               37 U10-LEDA2       53
6 R30-Q4           22 GND           38                 54
7 INS+             23 VQ5           39 MIC.C53-AIN     55 CL63-VQ5
8 INS-             24               40 MIC.TSC.AUX     56 R24-SR
9 VDET             25 VDD3.3        41 GND             57 SPRO
```

```
 10 PVDD           26 GND          42 R38-RED        58 SPLO
 11                27 CL60-VDD3.3  43 R37-GREEN      59 R22-SL
 12 PWSW           28 VSHD         44 VDD3.3         60 GND
 13 GND            29              45 PWM.SPI.CLK    61 R79-VR3.PIN2
 14 GND            30 VDD5         46 PWM.SPI.D      62
 15 GND            31 U9-LEDA1     47 PWM.SPI.Q      63
 16 VQ5            32              48 PWM.SPI.SEL    64
```

NDS-LITE Power Switch

```
 1 PWSW (grounded when switch is pulled)
 2 GND
 3 GND
 4 NC? (grounded when switch is not pulled)
```

Pinouts - Wifi

NDS RFU Daughter Board (Firmware FLASH, Wifi BB/RF Chips)

```
 1 N/A      6 FMW.CLK    11 ENABLE 16 RX.DTA? 21 BB./CS     26 22MHz   31 GND
 2 GND      7 FMW./SEL   12 GND    17 TX.MAIN 22 RF./CS     27 GND     32 GND
 3 high?    8 FMW.DTA.Q  13 GND    18 GND     23 BB.RF.CLK  28 VDD3.3  33 GND
 4 RXTX.ON  9 FMW.DTA.D  14 TX.ON  19 TX.CLK  24 BB.RF.RD   29 VDD1.8
 5 FMW./WP  10 FMW./RES  15 RX.ON  20 TX.DTA  25 BB.RF.WR   30 GND
```

NDS-Lite RFU Daughter Board DWM-W006 (Firmware FLASH, Wifi BB/RF Chip)

```
 1 GND      6 GND       11 BB.RF.WR  16 VDD3.3   21 RF.SLEEP  26 FMW.Q
 2 TXPE     7 TRCLK     12 BB.RF.CLK 17 GND      22 FMW./RES  27 FMW./WP
 3 RXPE     8 TRDATA    13 GND       18 RF./CS   23 GND       28 FMW./CS
 4 CCA      9 GND       14 MCLK      19 BB.SLEEP 24 FMW.CLK   29 LD  ;hi?
 5 TRRDY    10 BB.RF.RD 15 GND       20 BB./CS   25 FMW.D     30 GND
```

https://fccid.io/EW4DWMW006/Label/ID-label-format-and-location-706511.pdf

Wifi RF Chip: RF9008, 0441, E0121Q (32 pin)

```
 1       5       9       13      17          21 RF.CLK  25         29
 2       6       10      14 GND  18          22         26         30
 3       7       11      15      19 RF.RD    23         27         31
 4       8       12      16      20 RF./CS   24         28         32
```

Pin19 RF.RD (oops, should be WR, maybe I've exchanged RD-WR?)
Pin20 RF./CS (via 10ohm)
Pin21 RF.CLK (via 10ohm)
Resembles RF2958 datasheet...?

Wifi BB Chip: Mitsumi, Japan, 4418, MM3155 (48 pins)

```
 1 GND   7       13 GND     19      25       31         37 TX.MAIN 43
 2       8       14         20      26       32 BB./CS  38 RX.DTA? 44
 3       9       15 BB.CLK  21      27       33 TX.DTA  39 RX.ON   45 GND
 4       10      16 BB.WR   22      28 RST   34 RXTX.ON 40 TX.ON   46
 5       11      17 BB.RD   23      29       35 TX.CLK  41         47
 6       12      18 22MHz   24      30       36         42         48
```

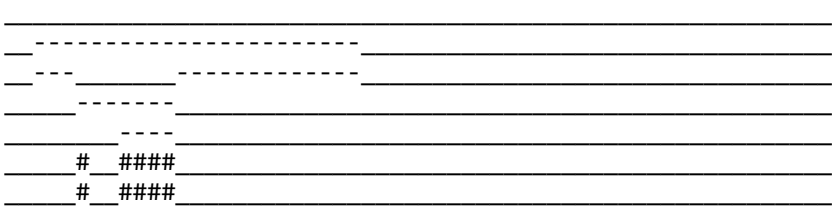
Pin15 BB.CLK (via 10ohm to RFU.23)
Pin16 BB.WR (RFU.25)
Pin17 BB.RD (RFU.24)
Pin18 22MHz (via 50ohm)
Pin28 RST (same as FMW./RES)
Pin32 BB./CS (RFU.21)
The chip is roughly resembling HSP3824/HFA3824A/HFA3860A/HFA3860B (though pinout and registers aren't compatible).

NDS-LITE BB/RF-Chip Mitsumi MM3218 (56 pins)

```
1-VDD18 8-GND   15-CCA    22-GND   29-VDD33     36-BBRF_SDO 43-VDD18 50-
2-GND   9-NC    16-TXPE   23-MCLK  30-          37-BB_SCS   44-...   51-
3-VDD18 10-GND  17-RXPE   24-VDD18 31-          38-RF_SCS   45-VDD33 52-VDD18
4-Ant   11-GND  18-TRRDY  25-NC    32-VDD18     39-GND      46-...   53-NC
5-Ant'  12-GND  19-TRDATA 26-22MHz 33-VDD18     40-VDD33    47-...   54-NC
6-VDD18 13-NC   20-VDD33  27-22MHz 34-BBRF_SCLK 41-RF_SLEEP 48-VDD33 55-NC
7-VDD18 14-/RES 21-TRCLK  28-      35-BBRF_SDI  42-BB_SLEEP 49-GND   56-NC
```

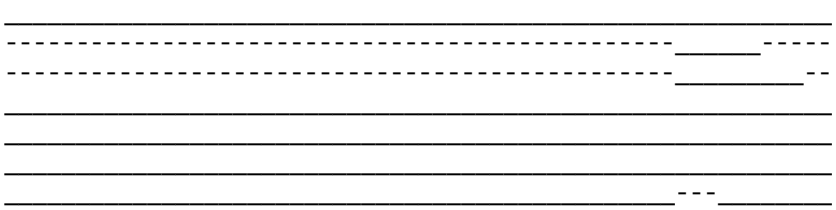
Note: Pinout should be same as in DSi (see DSi pinout for details).

TX Signal/Timing Chart (Host Game)



This example shows a host sending beacons. The pre-beacon receive period is probably to sense conflicts with other transmitters. The post-beacon receive period is to get responses from other players. The two transmit parts are: The hardware header, followed by inactivity on the tx pins during the rest of the preamble period, then followed by the actual IEEE frame. The rest of the time is spent in idle mode to reduce power consumption.

RX Signal/Timing Chart (Join Game)



This example shows a client trying to receive beacons, so most of the time is spent in receive mode (the short idle periods are probably occuring when it is switching to another channel). Once when it has associated with a host, the client may spend more time in idle mode, and needs to be in receive mode only when expecting to receive beacons or other data.

Pinouts - Various

Advance Gameboy 256Kbytes RAM 128Kx16bit (NEC D442012LGY-B85x-MJH) (wide)
GBA SP 256Kbytes RAM 128Kx16bit (F 82D12160-10FN) (square)

```
1 A15   7 A9    13 IC    19 A6   25 A0   31 D2   37 VCC  43 D15
2 A14   8 A8    14 /UB   20 A5   26 /CE1 32 D10  38 D5   44 D8
3 A13   9 NC    15 /LB   21 A4   27 GND  33 D3   39 D13  45 D16
4 A12   10 NC   16 NC    22 A3   28 /OE  34 D11  40 D6   46 GND
5 A11   11 /WE  17 NC    23 A2   29 D1   35 D4   41 D14  47 NC
6 A10   12 CE2  18 A7    24 A1   30 D9   36 D12  42 D7   48 A16
```

Connection in GBA and GBA SP: IC-GND, /CE1-GND, CE2-VDD2, VCC-VDD2, Pin16-VDD2, the other NC pins seem to be actually not connected, all other pins connect to the corresponding Wxx CPU pins. Note: Both GBA and GBA SP have soldering points for wide (12x18mm) and square (12x14mm) RAMs, so either could be used.
The GBA additionally contains 32K built-in WRAM, and built-in VRAM, so the above 256K RAM chip is probably not used in 8bit classic/color gameboy mode.
Note: In the GBA Micro, the 256K RAM are contained on-chip in the CPU.

Advance Gameboy Schematic Fragments
P2-VSS = VDD-15
VIN = VCC3 via R33
REGEXT (on my modified board, REGEXT underneath of my diodes)

/RES (OUT) (via R40)
/CS (via R39)
/WR (via R38)
SC (via Rxx)
SD (via Rxx)
SI (via Rxx)
SO (via Rxx)
DCK (via R36)
A-GND via CP4 (100uF) to GND (used speaker, and on headphone socket)

GBA SP Schematic Fragments
P2VDD = VDD13
P2VSS = VDD15
/RES via R46
/CS via R45
/WR via R44
DCK via R20
VS=BT+
In my repaired GBA-SP: CK1 test-point is disconnected (instead GND'ed).
In my repaired GBA-SP: broken oscillator replaced
In my repaired GBA-SP: broken r1 1mOhm replaced (near oscillator)
In my repaired GBA-SP: broken EXT2 socket metal-spring/snapper removed
CL1 FIQ (near SW4)
CL2 ?
CL3 ?
CL4 VOUT1/VDD1 (near U4)
CL5 VOUT3/VDD3 (near U4)
CL6 VOUT5/VDD5 (near U4)
DL1-red (power low) ---R32--Q4--R6--
DL2-green (power good) ---Q6--LOWBAT/R34-VDD3
DL3-orange (charge) --R24--Q2--VIN/U57
P2VDD--VDD13
P2VSS--VDD15
S+ and S- are (almost) shortcut by R23 (1.0 ohm)
S+ via Q1 to VIN
VS via D1 to S-
A-GND via CP1 (100uF) to GND
U4 pin 12 to r6 (towards red led)
U4 pin 14 to D6---to U7
SC (CPU pin48) with R7 100K ohm pullup to VDD35
P35 via Q11 to SW (speaker disable)

GBA SP Backlight-Button Schematic (U6,U8,Q12)

```
   ______                 _____
GND--|1 U8 6|-- U85        |     |--VDD5
U82--|2    5|-- U85    U61-| Q12 |         U83  ------> to display
U83--|3____4|-- U82        |_____|--Q12B   Q12B <------ from button
U61--|1 U6 8|--VDD5    (X)---R51--VDD5    (X)---C70--GND
U62--|2    7|--VDD5    U62---R49--VDD5    U61---R40--GND
U62--|3    6|--(X)     Q12B--R39--VDD5    U82---R38--GND
GND--|4____5|--NC?     Q12B--C69--VDD5    U85---R50--U62
```

AUX Xboo PC-to-GBA Multiboot Cable

Below describes how to connect a PC parallel port to the GBA link port, allowing to upload small programs (max 256 KBytes) from no$gba's Utility menu into real GBAs.

This is possible because the GBA BIOS includes a built-in function for downloading & executing program code even when no cartridge is inserted. The program is loaded to 2000000h and up in GBA memory, and must contain cartridge header information just as for normal ROM cartridges (nintendo logo, checksum, etc., plus some additional multiboot info).

Basic Cable Connection
The general connection is very simple (only needs four wires), the only problem is that you need a special GBA plug or otherwise need to solder wires directly to the GBA mainboard (see Examples below).

```
  GBA  Name  Color                SUBD CNTR Name
  2    SO    Red     ------------ 10   10   /ACK
  3    SI    Orange  ------------ 14   14   /AUTOLF
  5    SC    Green   ------------ 1    1    /STROBE
  6    GND   Blue    ------------ 19   19   GND
```

Optionally, also connect the following signals (see notes below):

```
  4    SD    Brown   ------------ 17   36   /SELECT  (double speed burst)
  3    SI    Orange  ----[===]---- 2..9 2..9 D0..7    (pull-up, 560 Ohm)
  5    SC    Green   ----[===]---- 2..9 2..9 D0..7    (pull-up, 560 Ohm)
  4    SD    Brown   ----[===]---- 2..9 2..9 D0..7    (pull-up, 560 Ohm)
  START  (mainboard) -----|>|----- 16   31   /INIT    (auto-reset, 1N4148)
  SELECT (mainboard) -----|>|----- 16   31   /INIT    (auto-reset, 1N4148)
  RESET  (mainboard) -----||------ 16   31   /INIT    (auto-reset, 300nF)
```

Notes: The GBA Pins are arranged from left to right as 2,4,6 in upper row, and 1,3,5 in lower row; outside view of GBA socket; flat side of socket upside. The above "Colors" are as used in most or all standard Nintendo link cables, note that Red/Orange will be exchanged at one end in cables with crossed SO/SI lines. At the PC side, use the SUBD pin numbers when connecting to a 25-pin SUBD plug, or CNTR pin numbers for 36-pin Centronics plug.

Optional SD Connection (Double Speed Burst)
The SD line is used for Double Speed Burst transfers only, in case that you are using a gameboy link plug for the connection, and if that plug does not have a SD-pin (such like from older 8bit gameboy cables), then you may leave out this connection. Burst Boot will then only work half as fast though.

Optional Pull-Ups (Improves Low-to-High Transition Speed)
If your parallel port works only with medium or slow delay settings, try to connect 560 Ohm resistors to SI/SC/SD inputs each, and the other resistor pin to any or all of the parallel port data lines (no$gba outputs high to pins 2..9).

Optional Reset Connection (CAUTION: Connection changed September 2004)
The Reset connection allows to automatically reset & upload data even if a program in the GBA has locked up (or if you've loaded a program that does not support nocash burst boot), without having to reset the GBA manually by switching it off and on (and without having to press Start+Select if a cartridge is inserted).
The two diodes should be 1N4148 or similar, the capacitor should be 300nF (eg. three 100nF capacitors in parallel). The signals are labeled on the mainboard, and can be found at following names / CPU pin numbers: RESET/CPU.125, SELECT/TP2/CPU.126, START/TP3/CPU.127.

Optional Power Supply Connection
Also, you may want to connect the power supply to parallel port data lines, see chapter Power Supply for details.

Transmission Speed
The first transfer will be very slow, and the GBA BIOS will display the boot logo for at least 4 seconds, even if the transfer has completed in less time. Once when you have uploaded a program with burst boot backdoor, further transfers will be ways faster. The table below shows transfer times for 0KByte - 256KByte files:

```
Boot Mode_____Delay 0_______Delay 1_______Delay 2_____
Double Burst  0.1s - 1.8s   0.1s - 3.7s   0.1s - 5.3s
Single Burst  0.1s - 3.6s   0.1s - 7.1s   0.1s - 10.6s
Normal Bios   4.0s - 9.0s   4.0s - 12.7s  4.0s - 16.3s
```

All timings measured on a 66MHz computer, best possible transmission speed should be 150KBytes/second. Timings might slightly vary depending on the CPU speed and/or operating system. Synchronization is done by I/O waitstates, that should work even on faster computers. Non-zero delays are eventually required for cables without pull-ups.

Requirements

Beside for the cable and plugs, no special requirements.
The cable should work with all parallel ports, including old-fashioned one-directional printer ports, as well as modern bi-directional EPP ports. Transfer timings should work stable regardless of the PCs CPU speed (see above though), and regardless of multitasking interruptions.
Both no$gba and the actual transmission procedure are using some 32bit code, so that either one currently requires 80386SX CPUs or above.

Connection Examples

As far as I can imagine, there are four possible methods how to connect the cable to the GBA. The first two methods don't require to open the GBA, and the other methods also allow to connect optional power supply and reset signal.

```
1) Connect it to the GBA link port. Advantage: No need to
   open/modify the GBA. Disadvantage: You need a special plug,
   (typically gained by removing it from a gameboy link cable).
2) Solder the cable directly to the GBA link port pins. Advantages:
   No plug required & no need to open the GBA. Disadvantages:
   You can't remove the cable, and the link port becomes unusable.
3) Solder the cable directly to the GBA mainboard. Advantage: No
   plug required at the GBA side. Disadvantage: You'll always
   have a cable leaping out of the GBA even when not using it,
   unless you put a small standard plug between GBA and cable.
4) Install a Centronics socket in the GBA (between power switch
   and headphone socket). Advantage: You can use a standard
   printer cable. Disadvantages: You need to cut a big hole into
   the GBAs battery box (which cannot be used anymore), the big
   cable might be a bit uncomfortable when holding the GBA.
```

Personally, I've decided to use the lastmost method as I don't like ending up with hundreds of special cables for different purposes, and asides, it's been fun to damage the GAB as much as possible.

Note

The above used PC parallel port signals are typically using 5V=HIGH while GBA link ports deal with 3V=HIGH. From my experiences, the different voltages do not cause communication problems (and do not damage the GBA and/or PC hardware), and after all real men don't care about a handful of volts, however, use at own risk.

AUX Xboo Flashcard Upload

Flashcard Upload

Allows to write data to flashcards which are plugged into GBA cartridge slot, cartridge is automatically started after writing. On initial power-up, hold down START+SELECT to prevent the GBA from booting the old program in the flashcard.
The Upload function in Utility menu uses flashcard mode for files bigger than 256KB (otherwise uses multiboot mode automatically). Also, there's a separate Upload to Flashcard function in Remote Access submenu, allowing to write files of 256KB or less to flashcard if that should be desired.

Supported Flashcards

Function currently tested with Visoly Flash Advance (FA) 256Mbit (32MB) Turbo cartridge. Should also work with older FA versions. Please let me know if you are using other flashcards which aren't yet supported.

Flashcard Performance
Writing to flashcards may become potentially slow because of chip erase/write times, cable transmission time, and the sheer size of larger ROM-images. However, developers whom are testing different builts of their project usually won't need to rewrite the complete flashcard, Xboo uses a highspeed checksum mechanism (16MB/sec) to determine which flashcard sector(s) have changed, and does then re-write only these sector(s).
To eliminate transmission time, data transfer takes place in the erase phases. Erase/write time depends on the flashcard type, should be circa 1-2 seconds per 256KB sector. Because the cartridge is programmed directly in the GBA there's no need to remove it from the GBA when writing to it.

Developers Advice
Locate your program fragments at fixed addresses, for example, code and data blocks each aligned to 64K memory boundaries, so that data remains at the same location even when the size of code changes. Fill any blank spaces by value FFh for faster write time. Reduce the size of your ROM-image by efficient memory use (except for above alignment trick). Include the burst boot backdoor in your program, allowing to re-write the flashcard directly without resetting the GBA.

Lamers Advice
Xboo Flashcard support does not mean to get lame & to drop normal multiboot support, if your program fits into 256KB then make it <both> flashcard <and> multiboot compatible - multiboot reduces upload time, increases your flashcard lifetime, and will also work for people whom don't own flashcards.

AUX Xboo Burst Boot Backdoor

When writing Xboo compatible programs, always include a burst boot "backdoor", this will allow yourself (and other people) to upload programs much faster as when using the normal GBA BIOS multiboot function. Aside from the improved transmission speed, there's no need to reset the GBA each time (eventually manually if you do not have reset connect), without having to press Start+Select (if cartridge inserted), and, most important, the time-consuming nintendo-logo intro is bypassed.

The Burst Boot Protocol
In your programs IRQ handler, add some code that watches out for burst boot IRQ requests. When sensing a burst boot request, download the actual boot procedure, and pass control to that procedure.

```
Send (PC)    Reply (GBA)
"BRST"       "BOOT"        ;request burst, and reply <prepared> for boot
<wait 1/16s> <process IRQ> ;long delay, allow slave to enter IRQ handler
llllllll     "OKAY"        ;send length in bytes, reply <ready> to boot
dddddddd     --------      ;send data in 32bit units, reply don't care
cccccccc     cccccccc      ;exchange crc (all data units added together)
```

Use normal mode, 32bit, external clock for all transfers. The received highspeed loader (currently approx. 180h bytes) is to be loaded to and started at 3000000h, which will then handle the actual download operation.

Below is an example program which works with multiboot, burstboot, and as normal rom/flashcard. The source can be assembled with a22i (the no$gba built-in assembler, see no$gba utility menu). When using other/mainstream assemblers, you'll eventually have to change some directives, convert numbers from NNNh into 0xNNN format, and define the origin somewhere in linker/makefile instead of in source code.

```
 .arm            ;select 32bit ARM instruction set
 .gba            ;indicate that it's a gameboy advance program
 .fix            ;automatically fix the cartridge header checksum
org 2000000h     ;origin in RAM for multiboot-cable/no$gba-cutdown programs
 ;-----------------
```

```
;cartridge header/multiboot header
 b     rom_start                ;-rom entry point
 dcb   ...insert logo here...   ;-nintento logo (156 bytes)
 dcb   'XBOO SAMPLE '           ;-title (12 bytes)
 dcb   0,0,0,0,  0,0            ;-game code (4 bytes), maker code (2 bytes)
 dcb   96h,0,0                  ;-fixed value 96h, main unit code, device type
 dcb   0,0,0,0,0,0,0            ;-reserved (7 bytes)
 dcb   0                        ;-software version number
 dcb   0                        ;-header checksum (set by .fix)
 dcb   0,0                      ;-reserved (2 bytes)
 b     ram_start                ;-multiboot ram entry point
 dcb   0,0                      ;-multiboot reserved bytes (destroyed by BIOS)
 dcb   0,0                      ;-blank padded (32bit alignment)
;------------------
irq_handler:  ;interrupt handler (note: r0-r3 are pushed by BIOS)
 mov    r1,4000000h             ;\get I/O base address,
 ldr    r0,[r1,200h] ;IE/IF     ; read IE and IF,
 and    r0,r0,r0,lsr 16         ; isolate occurred AND enabled irqs,
 add    r3,r1,200h   ;IF        ; and acknowledge these in IF
 strh   r0,[r3,2]               ;/
 ldrh   r3,[r1,-8]              ;\mix up with BIOS irq flags at 3007FF8h,
 orr    r3,r3,r0                ; aka mirrored at 3FFFFF8h, this is required
 strh   r3,[r1,-8]              ;/when using the (VBlank-)IntrWait functions
 and    r3,r0,80h ;IE/IF.7 SIO  ;\
 cmp    r3,80h                  ; check if it's a burst boot interrupt
 ldreq  r2,[r1,120h] ;SIODATA32 ; (if interrupt caused by serial transfer,
 ldreq  r3,[msg_brst]           ; and if received data is "BRST",
 cmpeq  r2,r3                   ; then jump to burst boot)
 beq    burst_boot              ;/
 ;... insert your own interrupt handler code here ...
 bx     lr                      ;-return to the BIOS interrupt handler
;------------------
burst_boot:     ;requires incoming r1=4000000h
 ;... if your program uses DMA, disable any active DMA transfers here ...
 ldr   r4,[msg_okay]            ;\
 bl    sio_transfer             ; receive transfer length/bytes & reply "OKAY"
 mov   r2,r0 ;len               ;/
 mov   r3,3000000h   ;dst       ;\
 mov   r4,0  ;crc               ;
@@lop:                          ;
 bl    sio_transfer             ; download burst loader to 3000000h and up
 stmia [r3]!,r0      ;dst       ;
 add   r4,r4,r0      ;crc       ;
 subs  r2,r2,4       ;len       ;
 bhi   @@lop                    ;/
 bl    sio_transfer             ;-send crc value to master
 b     3000000h  ;ARM state!    ;-launch actual transfer / start the loader
;------------------
sio_transfer:  ;serial transfer subroutine, 32bit normal mode, external clock
 str   r4,[r1,120h]  ;siodata32 ;-set reply/send data
 ldr   r0,[r1,128h]  ;siocnt    ;\
 orr   r0,r0,80h                ; activate slave transfer
 str   r0,[r1,128h]  ;siocnt    ;/
@@wait:                         ;\
 ldr   r0,[r1,128h]  ;siocnt    ; wait until transfer completed
 tst   r0,80h                   ;
 bne   @@wait                   ;/
 ldr   r0,[r1,120h]  ;siodata32 ;-get received data
 bx    lr
;---
msg_boot dcb 'BOOT'     ;\
msg_okay dcb "OKAY"     ; ID codes for the burstboot protocol
msg_brst dcb "BRST"     ;/
;------------------
download_rom_to_ram:
```

```
 mov  r0,8000000h  ;src/rom     ;\
 mov  r1,2000000h  ;dst/ram     ;
 mov  r2,40000h/16 ;length      ; transfer the ROM content
@@lop:                          ; into RAM (done in units of 4 words/16 bytes)
 ldmia [r0]!,r4,r5,r6,r7        ; currently fills whole 256K of RAM,
 stmia [r1]!,r4,r5,r6,r7        ; even though the proggy is smaller
 subs  r2,r2,1                  ;
 bne   @@lop                    ;/
 sub   r15,lr,8000000h-2000000h ;-return (retadr rom/8000XXXh -> ram/2000XXXh)
;-----------------
init_interrupts:
 mov  r4,4000000h               ;-base address for below I/O registers
 ldr  r0,=irq_handler           ;\install IRQ handler address
 str  r0,[r4,-4]   ;IRQ HANDLER ;/at 3FFFFFC aka 3007FFC
 mov  r0,0008h                  ;\enable generating vblank irqs
 strh r0,[r4,4h]   ;DISPSTAT    ;/
 mrs  r0,cpsr                   ;\
 bic  r0,r0,80h                 ; cpu interrupt enable (clear i-flag)
 msr  cpsr,r0                   ;/
 mov  r0,0                      ;\
 str  r0,[r4,134h] ;RCNT        ; init SIO normal mode, external clock,
 ldr  r0,=5080h                 ; 32bit, IRQ enable, transfer started
 str  r0,[r4,128h] ;SIOCNT      ; output "BOOT" (indicate burst boot prepared)
 ldr  r0,[msg_boot]             ;
 str  r0,[r4,120h] ;SIODATA32   ;/
 mov  r0,1                      ;\interrupt master enable
 str  r0,[r4,208h] ;IME=1       ;/
 mov  r0,81h                    ;\enable execution of vblank IRQs,
 str  r0,[r4,200h] ;IE=81h      ;/and of SIO IRQs (burst boot)
 bx   lr
;-----------------
rom_start:   ;entry point when booted from flashcart/rom
 bl   download_rom_to_ram       ;-download ROM to RAM (returns to ram_start)
ram_start:   ;entry point for multiboot/burstboot
 mov  r0,0feh                   ;\reset all registers, and clear all memory
 swi  10000h ;RegisterRamReset  ;/(except program code in wram at 2000000h)
 bl   init_interrupts           ;-install burst boot irq handler
 mov  r4,4000000h               ;\enable video,
 strh r4,[r4,000h] ;DISPCNT     ;/by clearing the forced blank bit
@@mainloop:
 swi  50000h ;VBlankIntrWait    ;-wait one frame (cpu in low power mode)
 mov  r5,5000000h               ;\increment the backdrop palette color
 str  r8,[r5]                   ; (ie. display a blinking screen)
 add  r8,r8,1                   ;/
 b    @@mainloop
;-----------------
.pool
end
```

DSi Emulation

Activating DSi emulation (disabled by default)
The DSi emulation is still DISABLED by default, and must be enabled manually by setting the "NDS Mode/Colors" option to "DSi (retail/16MB)".
The reason is that most DSi-enhanced games are still running into troubles in DSi emulation mode (but should work fine in NDS emulation mode).

DSi eMMC bootcode emulation
This is the newest feature, and also the most important breakthrough, as it's allowing to understand the boot process & memory initialization from scratch up. Getting there wasn't easy (eight months of research &

programming - just to get the emulation to display the damn bootscreen).
Using that emulation feature requires DSi BIOS dumps, eMMC dump, and Wifi-Flash dump. Getting that files together isn't exactly easy, but I hope that 5-10 people might known how to do it, and that 0-2 of them might be actually reading this someday.
The lower 32K halves of the ARM7/ARM9 BIOSes can be dumped quite easily (if you have any working DSi exploit), the DSi keys in upper 32K halves can be dumped indirectly (via simple exploits on 3DS, or via difficult main memory hacks on DSi). For details, see:
BIOS Dumping
Getting the eMMC image requires some soldering (or sudokuhax), the image must include a footer with CID/CPU IDs (needed for decryption; obviously those IDs will also tell Nintendo 'who you are', in case you should consider leaking your eMMC dump). For details, see:
DSi SD/MMC Images
DSi Console IDs
AUX DSi SD/MMC Pin-Outs
Optionally, a copy of the DSi Wifi Flash can be stored in file "WIFI-DSI.BIN" (should be usually 128Kbytes; or, reportedly it's smaller on newer DSi's). That file only contains only some stuff like wifi calibration values, no$gba is auto-generating some dummy calibration values if the file is missing (so the file is needed only if you want to do something like experimenting with different calibration settings).
If needed, the Wifi Flash can be dumped via SPI bus wiring (from wifi daughterboard), or via raw software (in case you can run NDS or DSi programs on your DSi console). For details, see the "Firmware" sections (named so because, on NDS, the Wifi Flash did also contain the firmware):
DS Cartridges, Encryption, Firmware
Then, set the "Reset/Startup Entrypoint" option in no$gba to "BIOS", and load a DSi cartridge or DSiware file (just as a dummy file, to switch the emulation into DSi mode; the file should be theoretically shown in bootmenu, but that doesn't work yet because the function of the tiny 4bit SCFG_MC register is still unknown).
Currently working is: The infamous health/safety crap, and displaying the bootmenu (selecting any items in bootmenu will start issuing eMMC writes, which isn't emulated yet, so the emu won't get any farther).

DSi Atheros Wifi BIOS
The Atheros Wifi BIOS is loaded from these filenames (in no$gba directory):

```
AR6002G.ROM  ;old DSi wifi chip (80Kbytes)   ;wifi-flash[1FDh]=1
AR6013G.ROM  ;new DSi wifi chip (256Kbytes)  ;wifi-flash[1FDh]=2
AR6014G.ROM  ;3DS wifi chip     (256Kbytes)  ;wifi-flash[1FDh]=3
```

presence of that file doesn't affect emulation, it's used solely for viewing the wifi bios in the disassembler (via menubar, window, xtensa/wifi). The wifi bios can be dumped via SDIO functions WINDOW_READ_ADDR and WINDOW_DATA.

DSi Cartridge emulation
For DSi ROM images, set the "Reset/Startup Entrypoint" to "Start Cartridge directly" (this will skip the eMMC boot process, so you don't need the eMMC dump, and you might also get away without BIOS dump, or with partial lower BIOS 32K dumps).
Then, load a DSi ROM-image (observe that older dumping tools can dump only the NDS areas of DSi cartridges; such incomplete dumps won't work in DSi mode).
Tested are two games: Cooking Coach (not working yet), and System Flaw (works, but requires cameras for use as "motion sensors", so it isn't actually playable). Going by reports from other users, most or all other DSi cartridges aren't working yet.

DSi DSiware emulation
DSiware games can be exported to SD card, and the SD card files can be decrypted via homebrew DSi SRL Extract utility (system files can be also downloaded and decrypted via homebrew NUS Downloader utility). Loading the decrypted files into no$gba is currently acting as if the files were cartridge ROM images (rather than files on eMMC). Surprisingly, this is actually working to some extent (the games are loading extra data from ROM, instead of from eMMC, which they should be usually doing). However, loading/saving game positions appears to be routed to eMMC (not to cartridge FLASH/EEPROM). Apart from that, most DSiware

games are probably stumbling over this or that unemulated detail or uninitialized memory area.

Nothing working?
Depends on what you want: If you want to play games, then, yes, I don't know of any DSi games that are working/playable yet. If you want to do programming/debugging, then you might find out that almost everything is more or less working. I don't know of any other active DSi devrs though - not too surprising since the DSi is definitely the most complicated and most unattractive console that I've ever seen.

Notes on New NDS Emulation

Hello to newer no$gba version(s) with NDS support. Most of it is working fine by now, a few details are still under construction. The most important (missing) things to be aware of are:

Major Missing Features
WLAN emulation is almost complete (but still incompatible with commercial games) (parts for transfer timing reasons, and parts because nobody knows yet how multiplayer packets work).
3D emulation is almost complete, too, only missing feature is Anti-Aliasing.

Things that work only with real NDS-BIOS-image
The "decompression-with-callback" BIOS functions are currently working only if you do have a copy of the real BIOSes as ROM-images (see Installation and DS Xboo chapters).
Encrypted ROM-images (with encrypted first 2Kbytes of the secure area) can be used only if the (full) NDS7 BIOS ROM-image is present (which contains the 1048h-byte decryption seed data).

Minor Missing Features and Glitches (in NDS mode)
VRAM viewer Tile/OBJ windows don't work.
Saving Snapshots doesn't work yet.
Backup media type isn't autodetected.

Recommended TCM Areas
No$gba decodes memory blocks by examining address bits24-27 (so memory is split into 16MB blocks). TCM emulation works fastest if DTCM and ITCM are each located at the bottom of free 16MB blocks (ie. blocks that do not contain other memory, such like WRAM, VRAM, etc.), recommended ITCM/DTCM base addresses would be:

```
ITCM at 00000000h, or (mirrored to) 01000000h.
DTCM at 01000000h, 0B000000h, 0C000000h, 0D000000h, or 0E000000h.
```

Other addresses are supported by no$gba v2.2c and up, for example, the Metroid Demo uses DTCM base overlapping with the Main Memory region, a possible advantage is that DTCM & Main RAM can be used as continous memory region, however, the emulation performance won't be optimal.

Note
Use Ctrl+T in debug mode to toggle between the two CPUs.

Pocketstation Emulation

The Sony Pocketstation is a memory card with ARM processor and 32x32 pixel LCD screen for Sony Playstation consoles. As such, it doesn't have much to do with GBA/NDS.
Anyways, as I haven't got around to implement ARM emulation in no$psx yet, the Pocketstation is currently emulated as part of the no$gba project (eventually it might be moving from no$gba to no$psx someday in future).

Pocketstation BIOS
For using a copy of the original 16Kbyte BIOS: store it as file POCKSTAT.ROM in no$gba directory. If that file isn't present, then no$gba will use it's own built-in BIOS clone.

Pocketstation Specs
For tech info on memory map, I/O ports, bios, file formats, see:
`http://problemkaputt.de/psx-spx.htm#pocketstation`
For info on the ARM processor, see:
ARM CPU Reference
For info on the POC-XBOO upload circuit see:
Pocketstation XBOO Cable

Pocketstation XBOO Cable

[This chapter will be moved to no$psx v1.7 specs; once when v1.7 is released]

This circuit allows to connect a pocketstation to PC parallel port, allowing to upload executables to real hardware, and also allowing to download TTY debug messages (particulary useful as the 32x32 pixel LCD screen is ways too small to display any detailed status info).

POC-XBOO Circuit
Use a standard parallel port cable (with 36pin centronics connector or 25pin DB connector) and then build a small adaptor like this:

```
Pin CNTR  DB25          Pocketstation          _____________________
ACK 10    10  --------- 1 JOYDTA              |       |       |     |
D0  2     2   --------- 2 JOYCMD              | 9 7 6 | 5 4 3 |  2 1  | CARD
GND 19-30 18-25 ------- 4 GND                |_______|_______|_______|
D1  3     3   --------- 6 /JOYSEL              _____________________
D2  4     4   --------- 7 JOYCLK             |       |       |     |
PE  12    12  --------- 9 /JOYACK (/IRQ7)     | 9 8 7 | 6 5 4 | 3 2 1 | PAD
NC -------------------- 8 /JOYGUN (/IRQ10)     \_______|_______|_____/
NC -------------------- 3 7.5V (rumble.supply)
SUPPLY.5V --|>|---|>|-- 5 3.5V (VCC) (eg. PC's +5V via two 1N4001 diodes)
SUPPLY.0V ------------- 4 GND (not needed when same as GND on CNTR/DB25)
```

The circuit is same as for "Direct Pad Pro" (but using a memory card connector instead of joypad connector, and needing +5V from PC power supply instead of using parallel port D3..D7 as supply). Note: IRQ7 is optional (for faster/early timeout).

POC-XBOO Upload
The upload function is found in no$gba "Utility" menu. It does upload the executable and autostart it via standard memory card/pocketstation commands (ie. it doesn't require any special transmission software installed on the pocketstation side).
Notes: Upload is overwriting ALL files on the memory card, and does then autostart the first file. Upload is done as "read and write only if different", this provides faster transfers and higher lifetime.

POC-XBOO TTY Debug Messages
TTY output is conventionally done by executing the ARM CPU's Undefined Opcode with an ASCII character in R0 register (for that purpose, the undef opcode handler should simply point to a MOVS PC,LR opcode).
That kind of TTY output works in no$gba's pocketstation emulation. It can be also used via no$gba's POC-XBOO cable, but requires some small customization in the executable:
First of, the executable needs "TTY+" ID in some reserved bytes of the title sector (telling the xboo uploader to stay in transmission mode and to keep checking for TTY messages after the actual upload):
`TitleSector[58h] = "TTY+"`
With that ID, and with the XBOO-hardware being used, the game will be started with with "TTY+" in R0 (notifying it that the XBOO hardware is present, and that it needs to install special transmission handlers):

```
;------------------
.data?
org  200h
...
tty_bufsiz equ 128  ;max=128=fastest (can be smaller if you are short of RAM)
func3_info:                                    ;\ ;\
 func3_buf_base     dd 0   ;fixed="func3_buf"     ;  ;  func3_info+00h
 func3_buf_len      dd 0   ;range=0..128          ;/ ;  func3_info+04h
 func3_stack        dd 0                             ;  func3_info+08h
 func3_buffer:      defs tty_bufsiz                  ;/ func3_info+0Ch
ptr_to_comflags     dd 0
...
.code
...
;------------------
tty_wrchr:   ;in: r0=char
 dd      0e6000010h ;=undef opcode            ;-Write chr(r0) to TTY
 bx      lr
;------------------
init_tty:  ;in: r0=param (from entrypoint)
 ldr     r1,=2B595454h ;"TTY+"                ;\check if xboo-cable present
 cmp     r1,r0                                ; (r0=incoming param from
 beq     @@tty_by_xboo_cable                  ;/executable's entrypoint)
;- - -
 mov     r1,0                                 ;\dummy und_handler
 ldr     r2,=0e1b0f00eh  ;=movs r15,r14       ; (just return from exception,
 str     r2,[r1,04h]  ;und_handler            ;/for normal cable-less mode)
 b       @@finish
;---
@@tty_by_xboo_cable:
 swi     17h  ;GetPtrToFunc3addr()            ;\
 ldr     r1,=(tty_func3_handler AND 0ffffh)   ; init FUNC3 aka TTY handler
 strh    r1,[r0]                              ;/
 ldr     r1,=func3_info                       ;\
 mov     r0,0                            ;\   ; mark TTY as len=empty
 str     r0,[r1,4]        ;func3_buf_len ;/   ; and
 add     r0,r1,0ch ;=func3_buffer        ;\   ; init func3 base
 str     r0,[r1,0]        ;func3_buf_base ;/  ;/
 mov     r1,0                                 ;\
 ldr     r2,=0e59ff018h  ;=ldr r15,[pc,NN]    ;
 str     r2,[r1,04h]  ;und_handler            ; special xboo und_handler
 add     r2,=tty_xboo_und_handler             ;
 str     r2,[r1,24h]  ;und_vector             ;/
@@finish:
 swi     06h ;GetPtrToComFlags()              ;\
 ldr     r1,=ptr_to_comflags                  ; get ptr to ComFlags
 str     r0,[r1]                              ;/
 bx      lr
;------------------
tty_xboo_und_handler:   ;in: r0=char
 ldr     r13,=func3_info ;aka sp_und          ;-base address (in sp_und)
 str     r12,[r13,8] ;func3_stack             ;-push r12
@@wait_if_buffer_full:                        ;\
 ldr     r12,=ptr_to_comflags                 ; ;\exit if execute file request
 ldr     r12,[r12]  ;ptr to ComFlags          ; ; ComFlg.Bit11 ("bu_cmd_59h"),
 ldr     r12,[r12]  ;read ComFlags            ; ; ie. allow that flag to be
 tst     r12,1 shl 11  ;test bit11            ; ; processed by main program,
 bne     @@exit                               ; ;/without hanging here
 ldrb    r12,[r13,4] ;func3_buf_len           ; wait if buffer full
 cmp     r12,tty_bufsiz                       ; (until drained by FIQ)
 beq     @@wait_if_buffer_full                ;/
 mov     r12,1bh+0c0h  ;mode=und, FIQ/IRQ=off ;\disable FIQ (no COMMUNICATION
 mov     cpsr_ctl,r12                         ;/interrupt during buffer write)
 ldrb    r12,[r13,4]  ;func3_buf_len          ;\
 add     r12,1        ;raise len              ; write char to buffer
```

```
  strb    r12,[r13,4]  ;func3_buf_len          ; and raise buffer length
  add     r12,0ch-1    ;=func3_buffer+INDEX    ;
  strb    r0,[r13,r12] ;append char to buf     ;/
 @@exit:
  ldr     r12,[r13,8]  ;func3_stack            ;-pop r12
  movs    r15,r14        ;return from exception (and restore old IRQ/FIQ state)
 ;------------------
 tty_func3_handler:   ;in: r0=flags, r1=ptr
  tst     r0,10h  ;test if PRE/POST data (pre: Z, post: NZ)
 ;ldreq   r1,[r1]   ;read 32bit param (aka the four LEN1 bytes of FUNC3)
  ldr     r0,=func3_info   ;ptr to two 32bit values (FUNC3 return value)
  movne   r1,0                          ;\for POST data: mark buffer empty
  strne   r1,[r0,4] ;func3_buf_len=0    ;/
  bx      lr                            ;-for PRE data: return r0=func3_info
```

Usage: Call "init_tty" at the executable's entrypoint (with incoming R0 passed on). Call "tty_wrchr" to output ASCII characters.
Note: The TTY messages are supported only in no$gba debug version (not no$gba gaming version).

Installation

Required Hardware
Miminum requirement would be a 32bit CPU (eg. 80386SX and up). No$gba has been programmed on a 66MHz 80486DX2, smaller games (eg. simple 2D games, text output) are working relative smoothly at 100% speed on that computer. Larger games (eg. 3D games, plasma demos) may require a CPU speed of up to 400..500MHz.
Emulation performance mainly depends on whether (and how much) the game is using the low power Halt/IntrWait GBA bios functions in idle mode, or not.

Installing No$gba
Just copy the executable to harddisk, preferably into a new/blank folder, below text will refer to this folder as the 'No$gba' folder, you may use any other name though. The program will automatically create some subdirecties. For example, the Slot directory is the suggested location for rom-images.
You will eventually also need some additional files - see below!

Installing Updates
Just overwrite the old executable by the newer version. The setup file from the older version may be used, any unrecognized options will be set to the default value.

Uninstalling No$gba
The program does not infect the windows registry. If you wish to uninstall it just delete the 'No$gba' folder and all sub-directories.
Of course excluding any important files which you may have stored in these directories and which you may want to keep (ie. source code, binaries, game positions, and/or your registration key).

Optional Files

NO$GBA.ROM
No$gba includes a built-in BIOS clone which is used if the BIOS ROM-image(s) are not found. However, the emulation accuracy (eg. exact execution time) of BIOS SWI-functions increases when using the original BIOS ROM-images. BIOS images are loaded if the following file(s) exist in your no$gba folder.

```
  NO$GBA.ROM    GBA+NDS7+NDS9 BIOS-images (16K+16K+4K)
```

Alternately, the images can be split into separate files, with filenames:

```
  BIOSNDS.ROM   NDS7+NDS9 BIOS-images (16K+4K)
  BIOSNDS7.ROM  NDS7 BIOS-image (16K)
  BIOSNDS9.ROM  NDS9 BIOS-image (4K) (or 32K padded with zero's)
```

```
  BIOSGBA.ROM   GBA BIOS-image (16K)
  GBA.ROM       Same as BIOSGBA.ROM (16K)
```
For DSi emulation,
```
  BIOSDSI.ROM   DSi7+DSi9 BIOS-images (64K+64K)  ;\these are only 66% dumpable
  BIOSDSI7.ROM  DSi7 BIOS-image (64K)            ; yet (please pad undumped
  BIOSDSI9.ROM  DSi9 BIOS-image (64K)            ;/regions with 00h-bytes)
```
For 3DS emulation,
```
  BIOS3DS.ROM   3ds9+3ds11 BIOS-images (64K+64K) ;\
  BIOS3DS9.ROM  3ds9 BIOS-image (64K)            ; these are filly dumpable
  BIOS3DS11.ROM 3ds11 BIOS-image (64K)           ;/
```
The no$gba utility menu (and the Xboo tool on the no$gba webpage) includes a function for downloading the BIOS from GBA/NDS to file (by simple cable connection).
Be careful when downloading the BIOSes from internet, there seems to be a very popular GBA BIOS in the net (probably a very old prototype version), which is NOT compatible with the actual GBA release version, also some NDS7 BIOS-images may be incomplete dumps (with first 4K zero-filled), no$gba outputs a warning message if using an incorrect, incomplete, damaged, or patched BIOS version.

FIRMWARE.BIN
A copy of the original NDS firmware. This file increases emulation accuracy only by means of delaying the boot process. The no$gba setup default setting is to start the cartridge directly, without executing the BIOS/Firmware boot code, so the presence of the firmware image doesn't actually disturb emulation performance.

DSi Firmware (onboard eMMC) and DSi External SD Card
DSi SD/MMC Images

PKUNZIP.EXE and/or ARJ.EXE
These decompression programs are required when loading ARJ or ZIP compressed cartridges. The programs must be stored in a directory which is part of your 'Path' (ie. a directory which is automatically searched by the operating system, if you do not understand this, try using your C:\Windows folder). Alternately, decompress the files by hand before loading them.

Debugging

Hotkeys in Debug Mode

Most debug functions are available via hotkeys and via mouse accessible popup boxes. The popup boxes generally also show the corresponding hotkey.

```
Cursor             (*) Move around
Cursor Right           Follow (in codewin: map window to jump/call dest adr)
Cursor Right           Follow (in stckwin: map codewin to return adr)
Cursor Left            Undo follow (if nothing to undo: goto program counter)
Page Up/Down       (*) Move around
Home               (*) Goto Start or to 0000
```

```
End                 (*) Goto End or to IO-Area (FF40 codewin, FF10 datawin)
Ret                 (*) Center/Uncenter current line in code window
Shift+Cursor            Change Active Window (Code,Data,Stck,Regs)
Shift+Cursor        (*) Toggle between Hex- and Ascii-input in data window
Tab                 (*) Toggle Standard and Symbolic Display in code window
Tab                 (*) Toggle Lower Window (Data or Break/Watch)
Ctrl-B                  Enter Breakpoint Address, Condition
Ctrl-N                  Find Next Breakpoint
Ctrl-G                  Goto Address (prompts for address) (does not affect pc)
Ctrl-E              (*) Toggle Warnings on/off
Ctrl-O                  OS Shell (calls DOS, type 'exit' to come back)
Ctrl-I                  Inspect (Define Watchpoint address)
Ctrl-R                  Reload Cartridge
Ctrl-S                  Search (see below! this works a bit strange)
Ctrl-C                  Continue Search
Ctrl-V             (**) Toggle Screen Size 25/50 lines (DOS version only)
Ctrl-D                  Toggle Datazone (see below)
Ctrl-A/T/X          (*) Add/Toggle/Remove Machine (up to 12 gameboys at 1 time)
Ctrl-L/W                Load/Save Snapshot (RAM, CPU-state and ROM-cartname)
Ctrl-Left/Right     (*) Decrease/Increase start address of window by one byte
<..>                    Assemble into Memory (input box appears on 1st char)
F1                      Help
F2                      Toggle Breakpoint at cursor
F3                      Trace with calls executed
F4                      Run to Cursor
F5                      VRAM Viewer (last accessed screen, TAB toggles)
F6                      Jump to Cursor (sets programcounter (pc) and rombank)
F7                      Trace (Execute one instruction)
F8                      Run until current sub-routine returns
F9                      Run (until breakpoint or user break)
F10                     Hardware Info Screen (splits in 50 lines DOS mode)
F11                     Setup Screen (last accessed setup window)
F12                     Cartridge Menu (last accessed, filename or files.lst)
Scroll Lock             Toggle Datacur follows Codecur (or 16bit reg) on/off
Keypad "/"              Run one Frame
Keypad "*"              Reset and Run
Keypad "-"          (*) Continue Run (same as F9)
ESC                 (*) Popup File Menu or close current window/menu
Alt+<..>            (*) Popup Menus (eg. Alt+F for File Menu)
Alt+A                   Animate (Repeated trace until key pressed)
Alt+B                   Place Bookmark
Alt+E                   Edit File
Alt+P                   Profiler Window
Alt+X                   Exit No$gba
Right Mouse Button  (*) DOS: Continue Run (same as F9), Windows: Context Menu
Left Mouse Button   (*) Select Window, Place Cursor, Toggle Breakpoint or
                         CPU-flag, Open Popup Menu, Click Option, etc.
```

The functions that are marked by (*) are not shown in the popup menues of the menu bar. Vice versa, not all functions can be accessed through hotkeys, so to be able to access all functions you must use both hotkeys and menu bars.

See also:
Hotkeys in Emulation Mode

Hotkeys in Emulation Mode

The default keyboard controls for player #1 and player #2 are

```
Up        Q          Cursor-Up
Down      A          Cursor-Down
Left      O          Cursor-Left
```

```
  Right     P          Cursor-Right
  Button A  Space      Keypad-"0"
  Button B  Tab        Keypad-"."
  Select    Left-Ctrl  Keypad-"3"
  Start     Enter      Keypad-Enter
  Button L  Left-Alt   L
  Button R  Right-Alt  R
```
In single player mode, both control sets do control the same gameboy.

The keyboard controls for player #1..#12 can be redefined in setup.
Note: The keyboard box in the DOS version wraps to the next page (for the next player) when you move the cursor out of the box with the cursor keys.

The other hotkeys during running emulation are:
```
  Alt+X                Exit No$gba
  Esc or Ctrl-Pause    Back to debugger (stop)
  keypad"-"            Back to debugger (start / stop)
  keypad"/"            Back to debugger (single-frame step)
  keypad"*"            Reset (Restart Game)
  keypad"+"            Whoosh. Run at max speed as long as key hold down
  Backspace            Whoosh. Same as above (for laptop's without keypad)
  Pause                Pause (Context Menu)
  Shift+F1..F5         Toggle Video Layers BG0..3,OBJ on/off
  F1..F4               Toggle Sound Channel 1..4 on/off
  F5..F6               Toggle Sound FIFO A..B on/off
  F7                   Quickload Snapshot (as 7 looks like L)
  F8                   Quicksave Snapshot (as 8 looks like S)
  F11                  Setup (Options)
  F12                  Cartridge menu
  both mouse buttons   Back to debugger
  left mouse button    Pause (menu) (under windows: right mouse button)
  other button(s)      Joypad emulation (fire/activate)
  mouse move           Joypad emulation (activated by first button click)
```

See Also:
Hotkeys in Debug Mode

Breakpoints

Normal Breaks (F2-key)
Normal breakpoints are set (or removed) by moving the code-window cursor onto the desired opcode, and then pushing the F2-key.

Run-to-Cursor (F4-key)
Hitting F4-key directly starts emulation, and stops when reaching the code window cursor. The break address is not memorized, ie. it is used only once.

Global Memory Read/Write Breaks
This break-family allows to capture reads/writes to specific memory addresses, or memory areas. Membreaks are defined by pressing Ctrl+B, and then entering a memory address or area in square brackets,
```
  [3007ffc]            single address (eg. GBA IRQ vector)
  [6000000..6003fff]   memory area    (eg. first 16K of VRAM)
```
followed by question and/or exclamation marks, indicating the type,
```
  ?     break on any read (from specified address/area)
  !?    break on any read or changed write
  !!?   break on any read or any write
  !     break on changed write
  !!    break on any write
```

Question marks ("?") capture reads. Double exclamation marks ("!!") will capture ALL writes, single exclamation marks ("!") capture only writes that are changing the memory content (ie. the new data must be different than old data). The ordering (eg. "!!?" or "!?!" or "?!!") is don't care.

Local Conditional Breakpoints
Press Ctrl-B and define the break by entering "address, condition". The emulator will stop when the program counter reaches the address, and when the condition is true. The "$" symbol represents the current cursor address of code window. Examples:

```
$, r0<>0           --> break at cursor position if r0 is non-zero
$, r0 & 2          --> break at cursor position if bit 1 of r0 is set
$, r0 !& 2         --> break at cursor position if bit 1 of r0 is zero
8001234, [r1]=r2   --> break at 8001234 if r1 points at a value equal to r2
wrchr              --> break at wrchr (always, no condition, same as F2-key)
wrchr, r0=0d       --> break at wrchr if r0 contains 0dh
$, [4000006] > 0A0 --> break at cursor if VCOUNT is greater than 0A0h
$, r4 <= r5        --> break at cursor if r4 is less or equal than r5
$, [r4] <> [r5]    --> break at cursor if r4 points at other value than r5
mainloop, ..5      --> break every 5th time that pc gets to mainloop (timer)
```

The conditions are verified BEFORE the instruction is executed.

```
Operators:              Operands:          Timer Identifier:
==  =   <   >   &       n   [nn]  r        ..
!=  <>  <=  >=  !&      nn  [rr]  rr
```

Operators == and != are pseudonyms for = and <>

Global Conditional Breakpoints
Global breaks are working exactly as above local breaks, except that the condition is verfied after <each> executed opcode. In the Ctrl+B window, enter a condition (without specifying a address). Examples:

```
r0 = 0            --> break whenever register r0 becomes zero
[4000006]>20      --> break whenever VCOUNT becomes greater than 20h
```

The emulator stops only when a condition changes from false to true, ie. it does not permanently break after each opcode if the condition stays true for a while.

The Break Window
The lower left window of the debug screen is shared for Data and Breakpoints, use TAB-key in this window to switch between Data and Break windows. The Break window lists all defined breakpoints, DEL-key can be used to delete a selected breakpoint. When pressing ENTER on a local breakpoint, the code window will be moved to the break address, this works also for bookmarks (that are non-functional 'dummy' breaks, defined by Alt+B key in code window).

Source Code Breakpoints
The opcode MOV R11,R11 may be used as source code breakpoint (in both THUMB and ARM state). These breaks are working much like normal F2-key breakpoints, except that they may be defined directly in source code by inserting the above opcode. Because the opcode is not changing any registers or flags there will be no conflicts when testing the binary on real hardware.
Note: Source breaks work only if you have enabled them in no$gba setup. To remove a specific source break, move the code-window cursor onto it, and hit the DEL-key (or enter NOP in the online assembler).

Profiling & Performance Monitoring

No$gba includes a couple of features for performance monitoring, allowing to write faster or smoother code, to free up more CPU time for additional effects, or simply to reduce the battery power consumption.

Profiling Features
Profiler Window
Profiler Compatibility

Clock Cycle Comments
Cycle Counters

Related Specifications...
ARM CPU Instruction Cycle Times
GBA Memory Map
GBA System Control
LCD Dimensions and Timings
BIOS Halt Functions

Profiler Window

The Profiler Window (Alt+P key in debug mode) displays execution time and number of calls of all executed procedures. The window is split into Tree, Path, and List tabs. Double-clicking an item in a tab moves the debuggers code window to the selected procedure.

Profiler Tree Tab
Displays all executed procedures with number of calls and execution time, presented as tree view; sub-routines show up as child branches. Useful to find out which procedures have been called how long and at which location(s).

Profiler Path Tab
Displays the current branch of the profiler tree - including any (nested) interrupt branch(es), the currently executed procedure shows up at the bottom. The path is similar to all return addresses on the various isr/irq/svc stacks (and in the LR register) being merged into a single "stack".

Profiler List Tab
Displays all executed procedures, the number of calls, and execution time in separate columns. The table can be sorted by clicking on the column headers.

Clock Cycle Values
The clock cycles can be optionally shown as total number of cycles (since the game was started, or the profiler list has been cleared), or as medium number of cycles per call, or as medium number of cycles per second (the GBA runs at 16,777,216 cycles per second).
Also, clock cycles can be either shown with or without cycles being spent in sub-routines (optionally including or excluding cycles in Halt/IntrWait sub-routines).

IRQs and DMAs
By default, IRQs and sound/video DMAs are shown as root entries in tree. If desired, the setup allows to change the profiler mode to count all IRQs/DMAs as "sub-routines". IRQs/DMAs which will then show up everywhere in the tree where they have occurred. Also, IRQs/DMAs clock cycles are then counted as normal sub-routine time to the interrupted "parent" procedures.
Note: DMAs that are directly started by the CPU (by start immediately timing) are always shown as local "sub-routines".

For more information on how to use the profiler, and how to avoid possible problems, please also read the next chapter,
Profiler Compatibility

Profiler Compatibility

The no$gba profiling mechanism allows the parent routines to include/exclude cycles being spent in sub-routines, and other useful effects like the tree tab which would be impossible with other less advanced profiling mechanisms.

How it works...
During emulation, the no$gba profiler keeps track of the program flow by watching any opcodes which change the program counter. It relies on identifying "call" and "return" opcodes used to enter/leave each procedure, and to ignore other "jump" opcodes used loop/skip inside of a procedure.
Sounds simple, but has been relative difficult to implement because the ARM CPU doesn't actually have clearly defined "call/return/jump" opcodes. Instead, it uses various branch (and load, pop, add, mov) opcodes to do various kinds of jumps to - or from - or inside of procedures (for example, the BX opcode is used for all three purposes).

Flexibility and possible problems
The profiler should be meanwhile quite flexible and should work with most programs. If it doesn't work, then it is probably only because of a few procedures which use unconvential program code. In that case it should be no big deal to solve the problem, either by changing that procedures, or by supporting it in no$gba. Either way, please let me know of <any> problems, and of which opcodes you are using to enter/leave the procedures, and if can solve it yourself by changing the code, or if you need the code to be supported in next no$gba update!

Re-synchronization
The profiler may shortly lose track of the program flow if it has overseen a return opcode from a child procedure to its parent. It will then assume that it is still executing the child (and any further cycles and sub-routines will be incorrectly counted to the child).
However, the profiler will re-synchronize itself to program flow once when it locates a return opcode which jumps to a higher parent procedure, the child routine which didn't have a return opcode will be then shown with a "(?)" question mark in the tree/list tabs.

Below is a summary of currently supported call/ret opcodes and opcode-combinations, useful to identify and report problems, or eventually to solve the problem by making your code compatible with it.

Calls - Entering a sub-routine
Sub-routines are most conventionally entered by a "bl addr" opcode. Calls with thumb/arm mode switches & indirect calls can be implemented by groups of two opcodes. In ARM mode, that'd be a "mov lr,pc" or a "add lr,=retadr" opcode, followed by a "bx rd" or "ldr pc,[jumplist]" opcode. Also, in ARM and THUMB mode, it can be implemented by a "bl addr" opcode which is pointing to a "bx rd" opcode. And, of course, IRQs and SWIs are as well supported.

Returns - Leaving a sub-routine
Standard returns would be "bx lr" or "mov pc,lr" or "pop pc" opcodes (aka "ldmfd [r13]!,pc" aka "ldr pc,[sp],4"). Also, any "bx rd" may be used for returns with thumb/arm mode switch (eg. after a thumb "pop rd"), or if the return address was moved to rd register (instead of saving it on stack). Note: The profiler identifies "bx rd" as return opcode if the current procedure (or any parent procdures) have been called with the same return address as the value in "rd". Finally, IRQs and SWIs may return by "subs/movs pc".

Jumps - Inside of procedures
All other jump opcodes are normally treated as normal jumps without affecting the profiler (unless a return address has been previously written into LR).
CAUTION: Because the normal thumb mode "b" opcode (branch) has relative short range, some programs may use the "bl" opcode (branch long with link) to produce long jumps (with the link register value being ignored/destroyed). The profiler currently treats all "bl" opcodes to be used to enter a sub-routine, not as long jump! Please let me know if that is a problem with your code!

LR - Link Register (R14)
Please always have the return address stored in LR when calling a procedure. And please avoid to use LR (R14) for other purposes, ie. if possible, use only R0-R12 as general purpose data registers.

Multitasking
The profiler memory is organized similar as a stack, and it should be actually running synchronous to the return addresses on the CPU stack. That means that it'd be currently not working with any 'multitasking' programs, ie. programs that switch between different threads, each with a separate stack.
Please let me know if you are doing any such things! I'd love to support it - if anybody wants it. The feature would probably work automatically (without you having anything much more to do than to say "hello, I use multitasking, what I am doing is..."), and it would nicely display each thread as separate root entry in the tree tab, allowing to determine time being spent in each thread (and its sub-routines) very easily...

Clock Cycle Comments

The normal execution time for one GBA opcode ranges between 1 and 100 (!) clock cycles, or more. Unfortunately, opcode execution time depends on various of circumstances (bus widths, waitstates, etc.), which is making it rather difficult to count cycles by hand, or just to a get feeling for efficient GBA code.

The no$gba Clock Cycle Comments feature displays the execution time for each opcode in disassembler window. Four different view modes (selectable in setup) are available:

Short Formula, and Detailed Formula
Short Formula: Displays the Non-sequential, Sequential, and Internal cycles (eg. 2N+3S+1I). Detailed Formula: Splits the N and S cycles into code and data access (eg. 1N+1S+1ND+2SD+1I, N/S for code, ND/SD for data).

Cycles, and Cycles & Sum
Cycles: Displays the number of clock cycles (by resolving the above formula - recursing operand size, opcode and data addresses, and current waitstate configuration). Cycles & Sum: Additionally displays the sum of all opcodes on screen in a second column (to use that feature, simply scroll the first line of interest to topmost screen position).

Even though these features are making it much easier to count cycles, there are still some restrictions which should be kept in mind:

Data access (load/store opcodes) - Important
Cycles are resolved by current register settings. For example, the execution time for "LDR Rd,[Rb]" could be resolved correctly only if Rb is currently pointing to the same memory area as when executing that opcode (that is valid if the program counter points to the opcode, for stack accesses by r13 register it'd be usually always valid).

Prefetch
Game-Pak Prefetch isn't yet understood or supported. If it is enabled in WAITCNT register, access to cartridge ROM may take up less cycles as displayed in clock cycle comments.

Conditional Opcodes
Conditional opcodes are always assumed to be true (unless if the program counter points to the opcode, in that case the actual execution time is displayed, ie. 1S if the condition is false).

Sums
The Cycles & Sums features simply adds one execution time to another, useful for fragments of code, but of course not aware of loops or skips.

Jumps
Execution time for branch opcodes is incorrectly displayed by using local waitstates for local code area (rather than target area). However, intersegment jumps (from one memory area to another) are typically consisting of a call-return combination (ie. the overall execution time would be correct, but the execution times of the call and return opcodes would be exchanged).

VRAM Waitstates
VRAM Waitstates are not displayed (the real hardware would occasionally add 1 cycle for accesses to video memory).

Cycle Counters

Execution Time value in Statusbar
No$gba displays the clock cycles difference between start and stop time in status bar when the emulation gets stopped. For example, hitting F3-key to execute a "BL draw_frame" opcode would display the execution time of the "draw_frame" procedure.
Note: That would also include any interrupts or DMAs which may have been occurred while executing that procedure.

User Counter
The user counter (located in Timers page of I/O map window) is incremented during emulation execution and tracing, the counter can be easily reset to zero by pushing the reset button which is displayed next to it.

Debug Message Counters
The Debug Message feature includes some special operands (%totalclks%, %lastclks%) which allow to write cycle counts to debug message window (or into log file). See Debug Messages chapter for details.

CPU Load Indicator
The "Power Gauge" (the red-green bar in debug window) displays the used CPU time (assuming that the unused CPU time is properly spent in Halt or IntrWait state, which reduces battery power consumption on real hardware). The higher it gets, the more CPU time is used.
By default, CPU load is re-calculated once per frame (60Hz), this rate can be changed in "Other" setup options (useful to prevent flickering with game engines that update the picture only once every two or three frames).
Note: The same information is also displayed as decimal value, in steps of 0.1%, in the Timers page of the I/O map window.

NB. Cycle counters are 32bit values and will overflow (restart at zero) after 4G cycles (256 seconds), which is probably more than enough in most cases.

Debug Messages

Debug Messages can be defined in source code. When the emulator 'executes' a message opcode, the message text will be output to the debug message window and to file DEBUGMSG.TXT. Debug Messages will not disturb execution on real hardware (the branch opcode just skips the data definition, wastes a few clock cycles though).

User Defined Messages
The message definition format (in THUMB or ARM mode) is as follows:

```
  mov  r12,r12          ;first ID
  b    @@continue       ;branch opcode that skips the message data
```

```
   dcw  6464h            ;second ID (ascii 'dd')
   dcw  0                ;reserved for flags
   dcb  'Hello'          ;user defined ascii data (max length 120 bytes)
   dcb  0                ;ending zero (normally not required, see below)
  .align 4               ;align following code (use align 2 in thumb mode)
  @@continue:
```

The text field is terminated by a zero byte or by the branch destination. The zero byte must be separately defined ONLY if your assembler doesn't zero-pad alignment space.

The ascii string may contain parameters, defined as %param%.

```
  r0,r1,r2,...,r15  show register content (displayed as 32bit Hex number)
  sp,lr,pc          alias for r13,r14,r15
  scanline          show current scanline number
  frame             show total number of frames since coldboot
  totalclks         show total number of clock cycles since coldboot
  lastclks          show number of cycles since previous lastclks (or zeroclks)
  zeroclks          resets the 'lastclks' counter
```

In the no$gba built-in A22i assembler, messages may be defined as:

```
  .msg 'Hello'
  .msg 'Source Addr = %r0% ; Dest Addr = %r1% ; Length = %r2%'
  .msg 'Stack Pointer = %SP%'
  .msg 'Decompression time: %lastclks% cycles'
```

Messages could be defined as macros when using other assemblers. The %parameters% are stored in ascii format as-is. The assembler/macro does NOT need to interprete and/or adjust the parameters!

See also

Aside from use "mov r12,r12", debug messages can be also output via pseudo I/O ports:
DS Debug Registers (Emulator/Devkits)

3D Log and Wifi Log

Additionally to the User messages, no$gba can also log 3D commands and Wifi packets in the debug message window. The separate message types can be enabled/disabled by checkboxes in the menubar of the window.

Symbolic Debug Info

If debug info (.SYM or .ELF files) has been loaded, labels will be displayed in disassembled code. And input boxes will recognize labels (eg. for Ctrl+G). TAB in code window toggles symbolic display on and off. Moving the code windows cursor onto a line with a symbol displays the value that hides behind the symbol in the upper left corner of the screen (DOS) or in the status bar (Windows).
Also, if source-level debug info is present, no$gba allows the user to view his/her source code in 'stacked' view mode, ie. disassembled opcodes shown below of each source line, this would be important for HLL programs.

.ELF Files (GBA)

Elf files are binaries, generated by many ARM assemblers and compilers (eg. Nintendo Tools, GNU tools). The files are containing the program (binary executable), and optionally also a symbol table & further debug information (usually in Dwarf2 format, and if present, typically containing source-level debug info).
Current no$gba version supports loading the binary (game), and the symbol table (similar content as .SYM files, but without additional code+data info), and the source-level debug information.
There seem to be different interpretations of how to store a binary inside of ELF files - as far as I know no$gba is compatible with all of these 'standards'.

.NEF/.SRL Files (NDS)

Nintendo's DS devkit outputs ROM-image and debug-info to separate files:

```
  cart.SRL rom-image (without any debug info, same as normal .NDS files)
  cart.NEF the NDS9 debug-info in ELF format (but without program code/data)
```

```
  cart.NLF contains a path to another .NEF file with NDS7 debug-info
```
No$gba v2.2e supports NDS9 debug info, the NDS7 file isn't supported yet.

.SYM Files
When not using ELF files, symbolic Information may be stored in file <cartname>.SYM in the same directory than <cartname>.GBA. The A22i assembler produces ready-for-use SYM files. When using another assembler, you may convert any debug information into .SYM format.

.SYM Symbolic Information File Format:
```
  ;no$gba symbolic information table example
  08000000 .arm
  080000C0 start
  08000124 mainloop
  080001EC .thumb
  080001EC init_video
  08000210 .arm
  08000210 irq_handler
  08000228 jumplist
  08000228 .dbl:0010
  08000414 text_array
  08000414 .asc:0017
  0800042B .asc:000F
  0800043A .asc:0012
  06000000 vram_base
  ;...
```

All symbols have to be declared as EIGHT-character string nnnnnnnn (hexadecimal memory address, not case sensitive), followed by at least one space or TAB and then followed by the symbol name (the name may contain any characters except spaces and control codes, names are case-sensitive).

.SYM Additional Code and Data Information
Aside from labels, the file may also contain information about 16bit or 32bt program code areas, and about data zones. This info is stored in the same format as normal labels, but by using the following reserved "labels":
```
  .arm          ;following code is in 32bit/ARM format
  .thumb        ;following code is in 16bit/THUMB format
  .byt:NNNN     ;next NNNN bytes are 8bit data (dcb lines)
  .wrd:NNNN     ;next NNNN bytes are 16bit data (dcw lines)
  .dbl:NNNN     ;next NNNN bytes are 32bit data (dcd lines)
  .asc:NNNN     ;next NNNN bytes are ascii data (quoted dcb lines)
  .pool         ;dummy label (indicates that following is literal pool)
```
The entries may be mixed with normal label definitions, NNNN is a hexadecimal value, it is always counted in BYTES (even when defining 16/32 bit data fields).

There is no need to sort the entries in any way. Empty lines are allowed. Lines starting with ";" are ignored as comments. Lines are terminated by LF or CRLF or EOF or filesize. The file is terminated by EOF or filesize.

XED Editor

XED About

About XED
XED is a text editor, the executable is fast and small, and it includes comfortable editing functions. It is both intended for standard .TXT files (or any other ASCII files, such like .ASM for assembler source code). Also, the line-wrapping support (.XED files) can be used for authoring stories, specifications, etc. Most of the features are much the same as for other text editors, some special functions are pointed out below:

Block Selection
XED supports good old quality block mechanisms, allowing to copy/move the selection directly to cursor position by Ctrl+K,C/V hotkeys (without needing to use paste). For data exchange with other programs or text files, the block can be directly written to (or loaded from) file by Ctrl+K,R/W. And, mainstream copy/cut/paste functions are supported as well, by Ctrl+Ins, Shift+Del, Shift+Ins.
Note: The block remains selected even when typing text, and it won't get deleted when touching Del-key.

Condensed Display Mode
Condensed mode is activated by "F6,C" key combination. In this mode, only lines beginning with colon ":", or (for assembler source code files) with semicolon-colon ";:", for example:

```
:Chapter IV
;:---Sound Engine---
```

Normal block functions can be used in this mode to Move, Copy, or Delete whole 'chapter(s)'. Cursor keys can be used to move the cursor to a specific chapter. Pushing Enter or Escape terminates condensed mode.

Column Block Mode
Column mode is activated by "Ctrl+K,N" key combination. In this mode, the block selection appears as a rectangular area, allowing to deal with tables & columns in text files by using copy/delete, indent/unindent block functions.
Typing "Ctrl+K,N" again will return to normal block mode (in which any lines between begin/end of the block will be selected at full length).

Blank Space
Unlike most other editors, XED allows to move the cursor to any screen location, including at horizontal positions after the end of the current line. Entering space characters at such locations advances the cursor position, but does not actually store space characters in the file.
When typing text, spaces are automatically inserted between line-end and cursor position. Respectively, ending spaces are automatically deleted (eg. assume that the line contains "Hello !", deleting "!" will also remove the space character, internally).
That is of course all happening behind your back, you won't have to care about it - but you can be sure that there'll be no hidden spaces filling up disk space.

Tabulation Marks / TAB-Key
The TAB Key advances the cursor to the next higher tabulation location (usually in steps of eight columns, counted from leftmost screen border), and the appropriate number of spaces is inserted into the file if necessary. In overwrite mode (de-/activated by INS Key), the TAB Key simply advances the cursor without actually inserting spaces (and without overwriting existing text by spaces).

Tabulation Marks / CHR(9)
When enabled in setup (default), TAB marks are automatically expanded into appropriate number of spaces (ie. towards next "8-column" position) when loading a file.
The file is normally saved by using raw SPC characters, without any TABs. Optionally, it can be saved by using "best-fill" SPCs and TABs (disabled by default), that feature may conflict with third party tools (assemblers, compilers, etc). In order to reduce the risk of such problems, best-fill is suppressed in quoted lines (by using ' or " or <> quotation marks, eg. db 'Hello !').

Line Wrapping

Line wrapping is enabled/disabled by "F5+W" key combination. Wrapping is automatically enabled when loading a file with extension ".XED".
In the file, wrapped lines are using CR characters as soft linebreaks, paragraphs are terminated by normal CR,LF characters.
Note: It'd be recommended to convert .XED files into 'standard' formats such like .TXT or .HTM before releasing them, but preferably NOT into disliked bloated file formats such like .PDF or .DOC.

Word Processing
Aside from the above line-wrapping support, no other 'word processing' features are included, the program provides normal 'type writer' functions, not more, not less. In particular, any overload such like bold or colored text, big and small fonts, bitmaps and different fonts are disliked.

XED Hotkeys

XED recognizes both CP/M Wordstar hotkeys (also used by Borland PC compilers), and Norton editor hotkeys (NU.EXE). The "Ctrl+X,Y" style hotkeys are wordstar based, particulary including good block functions. The F4,X and Alt/Ctrl+X type hotkeys are norton based, particulary very useful for forwards/backwards searching.

Standard Cursor Keys

Up	Move line up
Down	Move line down
Left	Move character left
Right	Move character right
Pgup	Scroll page up / to top of screen
Pgdn	Scroll page down / to bottom of screen
Ctrl+Pgup	Go to start of file (or Ctrl+Home)
Ctrl+Pgdn	Go to end of file (or Ctrl+End)
Home	Go to start of line
End	Go to end of line
Ctrl+Left	Move word left
Ctrl+Right	Move word right
Ins	Toggle Insert/Overwrite mode
Del	Delete char below cursor
Backspace	Delete char left of cursor
Tab	Move to next tabulation mark
Enter	New line/paragraph end
Esc	Quit (or Alt+X, F3+Q, Ctrl+K+D, Ctrl+K+Q, Ctrl+K+X)

Note: Pgup/Pgdn are moving the cursor to top/bottom of screen, page scrolling takes place only if the cursor was already at that location.

Editor Keys

Ctrl+Y	Delete line (or Alt+K)
Alt+L	Delete to line end (or Ctrl+Q,Y)
Alt+V	Caseflip to line end
Ctrl+V	Caseflip from line beginning

Norton Search/Replace Functions

Alt+F	Norton - search/replace, forwards
Ctrl+F	Norton - search/replace, backwards
Alt+C	Norton - continue search/replace, forwards
Ctrl+C	Norton - continue search/replace, backwards

Search: Type "Alt/Ctrl+F, String, Enter".
Search+replace: "Type Alt/Ctrl+F, String1, Alt+F, String2, Enter".
Non-case sensitive: Terminate by Escape instead of Enter.

Wordstar Search/Replace Functions

Ctrl+Q,F	Wordstar - search

```
  Ctrl+Q,A   Wordstar - replace
  Ctrl+L     Wordstar - continue search/replace
```
Search options: B=Backwards, G=Global, N=No query,
U=non-casesensitive, W=whole words only, n=n times.

Disk Commands
```
  F3,E       Save+exit
  F3,S       Save (or Ctrl+K,S)
  F3,N       Edit new file
  F3,A       Append a file
```
See also: Block commands (read/write block).

Block Selection
```
  Shift+Cursor  Select block begin..end
  Ctrl+K,B   Set block begin (or F4,S)
  Ctrl+K,K   Set block end   (or F4,S)
  Ctrl+K,H   Remove/hide block markers (or F4,R)
  F4,L       Mark line including ending CRLF (or Ctrl+K,L)
  F4,E       Mark line excluding ending CRLF
  Ctrl+K,T   Mark word
  Ctrl+K,N   Toggle normal/column blocktype
```

Clipboard Commands
```
  Shift+Ins  Paste from Clipboard
  Shift+Del  Cut to Clipboard
  Ctrl+Ins   Copy to Clipboard
  Ctrl+Del   Delete Block
```

Block Commands
```
  Ctrl+K,C   Copy block (or F4,C)
  Ctrl+K,V   Move block (or F4,M)
  Ctrl+K,Y   Delete block (or F4,D)
  Ctrl+K,P   Print block  (or F7,B)
  Ctrl+Q,B   Find block begin (or F4,F)
  Ctrl+Q,K   Find block end   (or F4,F)
  Ctrl+K,R   Read block from disk towards cursor location
  Ctrl+K,W   Write block to disk
  Ctrl+K,U   Unindent block (delete one space at begin of each line)
  Ctrl+K,I   Indent block (insert one space at begin of each line)
  F5,F       Format block (with actual x-wrap size) (or ;Ctrl+B)
  F8,A       Add values within column-block
```

Setup Commands
```
  F11        Setup menu (or F8,S)
  F5,S       Save editor configuration
  F5,L       Set line len for word wrap (or Ctrl+O,R)
  F5,W       Wordwrap on/off (or Ctrl+O,W) (*)
  F5,I       Auto indent on/off (or Ctrl+O,I)
  F5,T       Set tab display spacing
```
(*) Wrapped lines will be terminated by CR, paragraphs by CRLF.

Other
```
  F1         Help
  F2         Status (displays info about file & currently selected block)
  F8,M       Make best fill tabs
  F8,T       Translate all tabs to spaces
  SrcLock    Freeze cursor when typing text ("useful" for backwards writing)
  Ctrl+O,C   Center current line
  Ctrl+K,#   Set marker (#=0..9)
  Ctrl+Q,#   Move to marker (#=0..9)
  Ctrl+Q,P   Move to previous pos
```

```
  F6,C       Condensed display mode on/off (*)
  Ctrl+G     Go to line nnnn (or F6,G) (or commandline switch /l:nnnn)
```
(*) only lines starting with ':' or ';:' will be displayed. cursor and block commands can be used (e.g. to copy a text-sequence by just marking it's headline)

Hex-Edit Keys (Binary Files)
This mode is activated by /b commandline switch, allowing to view and modify binary files. Aside from normal cursor keys, the following hotkeys are used:
```
  Tab        Toggle between HEX and ASC mode (or Shift+Left/Right)
  Ctrl+Arrow Step left/right one full byte (instead one single HEX digit)
  Ctrl+G     Goto hex-address
  Ctrl+K,S   Save file (as usually)
```

Printer Commands
```
  F7,P       Print file
  F7,B       Print block (or Ctrl+K,P)
  F7,E       Eject page
  F7,S       Set page size
```
More printer options can be found in setup. Printing was working well (at least with my own printer) in older XED versions, but it is probably badly bugged (at least untested) for years now.

XED Assembler/Debugger Interface

Nocash Debuggers
The XED editor provides simple but very useful interaction with the various nocash debuggers/emulators (nogba, nogmb, nocpc, nomsx, no$c64, no$2k6, nozx, nones, nosns, nox51).
The editor can be launched from inside of the debugger (by Alt+E hotkey, by retaining the recently edited line number when re-launching the editor).
And, when editing assembler source code files, F9-key can used to launch the assembler from inside of XED. That is, the file is saved to disk, the A22i assembler is started (built-in in all debuggers), and, in case of successful assembly, the program is loaded & started in the emulator. Otherwise, the assembler displays a list of errors, and the editor is moved directly to the source code line number in which the first error has occured.

16bit DOS debuggers
The XED editor is included built-in in all nocash windows debuggers, and in the no$gba 32bit DOS version only.
For use with other nocash 16bit DOS debuggers the XED editor must be downloaded separately at http://problemkaputt.de/xed.htm, and must be installed in a directory which is included in your PATH statement.

XED Commandline based standalone version

Standalone 16bit DOS version
This version is written in turbo pascal, nevertheless fast enough to work on computer with less than 10MHz. It uses 16bit 8086 code, and works with all 80x86 compatible CPUs, including very old XTs.
The downside is that it is restricted to Conventional DOS Memory, so that the maximum filesize is 640K (actually less, because the program and operating system need to use some of that memory).

Using the 32bit debugger-built-in version as 32bit standalone editor
I haven't yet compiled a 32bit standalone version, however, any of the no$xxx 32bit debuggers can be used for that purpose. By commandline input:
```
  no$xxx /x <filename>   Edit text file in standalone mode
  no$xxx /b <filename>   Edit binary file in standalone hexedit mode
```

Standalone Commandline Syntax
Syntax: XED <filename> [/l:<line number>] | /?

```
  <name>    Filename, optionally d:\path\name.ext
  /?        Displays commandline help
  /l:<nnn>  Moves to line number nnn after loading
```

The filename does not require to include an extension, the program automatically loads the first existing file with any of following extensions appended: XED, ASM, ASC, INC, BAT, TXT, HTM, DOC, A22, PAS.

Standalone DOS Return Value
XED returns a three-byte return value after closing the program. This data is used when calling XED as external editor from inside of nocash DOS debuggers, but it might be also useful for other purposes.
Because normal DOS return values are limited to 8bits only, the three bytes are written into video RAM at rightmost three character locations in first line:

```
  VSEG:77*2  Exit code   (00h=Exit normal, F9h=Exit by F9-key)
  VSEG:78*2  Line number (Lower 8bits, 1..65536 in total)
  VSEG:79*2  Line number (Upper 8bits)
```

The color attribute for these characters is set to zero (invisible, black on black). Use INT 10h/AH=0Fh to determine the current video mode (AL AND 7Fh), if it is monochrome (07h) then use VSEG=B000h, otherwise VSEG=B800h.

The A22i Assembler

Online Assembler
In the debuggers code window, you can directly enter assembler instructions and modify the program code. The assembler input box shows up automatically when typing the first character of the new instruction.
The most basic purpose would be to overwrite an opcode by typing "NOP", but you could also enter any other instruction, such like "ADD R0,R0,4" when having debug info (.sym file) loaded, you may also use labels as parameters, eg. "B mainloop". The pseudo opcode "LDR Rd,=Imm" can be used only under certain circumstances:
Using LDR Rd,=Imm in Online Assembler
Note: Aside from translating machine code instructions, the assembler input box can be also used to change the value of CPU registers, by typing "r0=12345678", or "sp=3007FF0" for example. For curiosity, it also parses simple commands such like: run, reload, help, exit, date, etc.

Source Code Assembler
The assembler can be also used to assemble source code files, by using the Assemble File function in Utility menu. This function has been mostly implemented for my own pleasure, and because I didn't wanted to install thirdparty tools on my computer. It's having some nice features, but it's also lacking some possibly important functions (see below).

A22i Advantages
It's easy to use, very fast, produces ready-for-use debug info .sym files, it understands official ARM syntax, as well as more reasonable cleaned-up syntax, and also different assembler dialects, directives, and numeric formats, it can be used directly from inside of the debugger (and from inside of the editor), and vice-versa: it's able call the editor in case of assembly errors.

A22i Disadvantages
The main disadvantage is that A22i isn't supporting macros (yet).

A22i is a Matter of Taste
If you want to use linkers, make files, project files, header files, command files, resource files, script files,

libraries, high level languages, strip commands, object files, and cathode ray tube (crt) files - forget about it! A22i translates a source code file into a binary file, not more, not less.

For more info about instruction set and directives, see:
ARM CPU Reference
ARM Pseudo Instructions and Directives

Sample Code
For a source code example, have a look at the Magic Floor game at the no$gba webpage, the package includes the binary, source code, with comments, and .sym debug info file.
http://problemkaputt.de/magicflr.htm

Using LDR Rd,=Imm in Online Assembler

LDR{cond} Rd,=Imm is a pseudo opcode, consisting of a LDR Rd,[PC,disp] opcode, and a 32bit immediate at [PC+disp] in the literal pool. The pool is normally generated when assembling the source code, however, additional pool values can be eventually squeezed into the final binary:

1) If the cursor points to an existing LDR=IMM, then it can be overwritten it by a new LDR=IMM, with changed {cond}, changed Rd, or changed Imm. In the latter case, if the immediates pool entry is shared with other LDR=IMMs, then the value of those opcodes will get changed as well - if you wish to suppress that behaviour: First overwrite the old LDR=IMM by NOP, then overwrite it again by new LDR=IMM, which will then (try to) use below mechanisms...

2) If the cursor does NOT point to an existing LDR=IMM, then no$gba will search nearby memory for usable addresses - A) a 32bit value that matches the desired immediate, B) a word-aligned 32bit value containing ascii letters "fREE", which you may want to insert here and there in your code, or C) a presumably unused blank memory area of at least 256 bytes which is 00h- or FFh-filled.

3) If the debugged program was written in C or C++ then you can probably replace some of the meaningless opcodes by hand. For example, to load a 16bit immediate with ARM code: MOV Rd,LSB, ADD Rd,MSB.

About this Document

About
GBATEK written 2001-2014 by Martin Korth, programming specs for the GBA and NDS hardware, I've been trying to keep the specs both as short as possible, and as complete as possible. The document is part of the no$gba debuggers built-in help text.

Updates
The standalone docs in TXT and HTM format are updated when having added any major changes to the document. The no$gba built-in version will be updated more regularly, including for minor changes, along with all no$gba updates.

Homepage
http://problemkaputt.de/gba.htm - no$gba emulator homepage (freeware)
http://problemkaputt.de/gba-dev.htm - no$gba debugger homepage
http://problemkaputt.de/gbapics.htm - no$gba debugger screenshots
http://problemkaputt.de/gbatek.htm - gbatek html version
http://problemkaputt.de/gbatek.txt - gbatek text version

Feedback
If you find any information in this document to be misleading, incomplete, or incorrect, please say something!
My spam-shielded email address is found at:
http://problemkaputt.de/email.htm - contact
Mail from programmers only, please. No gaming questions, thanks.

Credits
Thanks for GBATEK fixes, and for info about GBA and NDS hardware,
- Jasper Vijn
- Remi Veilleux (DS video details)
- Randy Linden
- Sebastian Rasmussen
- Stephen Stair (DS Wifi)
- Cue (DS Firmware bits and bytes)
- Tim Seidel (DS Wifi RF2958 datasheet)
- Damien Good (DS Bios Dumping, and lots of e-Reader info)
- Kenobi and Dualscreenman (lots of ARDS/CBDS cheat info)
- Flubba (GBA X/Y-Axis tilt sensor, and GBA Gameboy Player info)
- DarkFader (DS Key2)
- Dstek by neimod (DS Sound)
- Christian Auby
- Jeff Frohwein
- NDSTech Wiki, http://www.bottledlight.com/ds/ (lots of DS info)
- wwylele (xpertteak cpu and mmio findings)

Formatting
TXT is 80 columns, TXT is 80 columns, TXT is 80 columns.
Don't trust anything else. Never.

Index

[extracted from no$gba v3.02 documentation]