Contents

fullsnes - nocash SNES hardware specifications, extracted from no$sns v1.5

SNES I/O Map

First some bytes

```
  0000h..1FFFh - WRAM - Mirror of first 8Kbyte of WRAM (at 7E0000h-7E1FFFh)
  2000h..20FFh - N/A  - Unused
```

PPU Picture Processing Unit (Write-Only Ports)

```
  2100h - INIDISP - Display Control 1                              8xh
  2101h - OBSEL   - Object Size and Object Base                    (?)
  2102h - OAMADDL - OAM Address (lower 8bit)                       (?)
  2103h - OAMADDH - OAM Address (upper 1bit) and Priority Rotation (?)
  2104h - OAMDATA - OAM Data Write (write-twice)                   (?)
  2105h - BGMODE  - BG Mode and BG Character Size                  (xFh)
  2106h - MOSAIC  - Mosaic Size and Mosaic Enable                  (?)
  2107h - BG1SC   - BG1 Screen Base and Screen Size                (?)
  2108h - BG2SC   - BG2 Screen Base and Screen Size                (?)
  2109h - BG3SC   - BG3 Screen Base and Screen Size                (?)
  210Ah - BG4SC   - BG4 Screen Base and Screen Size                (?)
  210Bh - BG12NBA - BG Character Data Area Designation             (?)
  210Ch - BG34NBA - BG Character Data Area Designation             (?)
  210Dh - BG1HOFS - BG1 Horizontal Scroll (X) (write-twice) / M7HOFS   (?,?)
  210Eh - BG1VOFS - BG1 Vertical Scroll (Y)   (write-twice) / M7VOFS   (?,?)
  210Fh - BG2HOFS - BG2 Horizontal Scroll (X) (write-twice)        (?,?)
  2110h - BG2VOFS - BG2 Vertical Scroll (Y)   (write-twice)        (?,?)
  2111h - BG3HOFS - BG3 Horizontal Scroll (X) (write-twice)        (?,?)
  2112h - BG3VOFS - BG3 Vertical Scroll (Y)   (write-twice)        (?,?)
  2113h - BG4HOFS - BG4 Horizontal Scroll (X) (write-twice)        (?,?)
  2114h - BG4VOFS - BG4 Vertical Scroll (Y)   (write-twice)        (?,?)
  2115h - VMAIN   - VRAM Address Increment Mode                    (?Fh)
  2116h - VMADDL  - VRAM Address (lower 8bit)                      (?)
  2117h - VMADDH  - VRAM Address (upper 8bit)                      (?)
  2118h - VMDATAL - VRAM Data Write (lower 8bit)                   (?)
  2119h - VMDATAH - VRAM Data Write (upper 8bit)                   (?)
  211Ah - M7SEL   - Rotation/Scaling Mode Settings                 (?)
  211Bh - M7A     - Rotation/Scaling Parameter A & Maths 16bit operand(FFh)(w2)
  211Ch - M7B     - Rotation/Scaling Parameter B & Maths 8bit operand (FFh)(w2)
  211Dh - M7C     - Rotation/Scaling Parameter C         (write-twice) (?)
  211Eh - M7D     - Rotation/Scaling Parameter D         (write-twice) (?)
  211Fh - M7X     - Rotation/Scaling Center Coordinate X (write-twice) (?)
  2120h - M7Y     - Rotation/Scaling Center Coordinate Y (write-twice) (?)
  2121h - CGADD   - Palette CGRAM Address                          (?)
  2122h - CGDATA  - Palette CGRAM Data Write             (write-twice) (?)
  2123h - W12SEL  - Window BG1/BG2 Mask Settings                   (?)
  2124h - W34SEL  - Window BG3/BG4 Mask Settings                   (?)
  2125h - WOBJSEL - Window OBJ/MATH Mask Settings                  (?)
  2126h - WH0     - Window 1 Left Position (X1)                    (?)
  2127h - WH1     - Window 1 Right Position (X2)                   (?)
  2128h - WH2     - Window 2 Left Position (X1)                    (?)
  2129h - WH3     - Window 2 Right Position (X2)                   (?)
  212Ah - WBGLOG  - Window 1/2 Mask Logic (BG1-BG4)                (?)
  212Bh - WOBJLOG - Window 1/2 Mask Logic (OBJ/MATH)               (?)
  212Ch - TM      - Main Screen Designation                        (?)
  212Dh - TS      - Sub Screen Designation                         (?)
  212Eh - TMW     - Window Area Main Screen Disable                (?)
  212Fh - TSW     - Window Area Sub Screen Disable                 (?)
  2130h - CGWSEL  - Color Math Control Register A                  (?)
  2131h - CGADSUB - Color Math Control Register B                  (?)
  2132h - COLDATA - Color Math Sub Screen Backdrop Color           (?)
  2133h - SETINI  - Display Control 2                              00h?
```

PPU Picture Processing Unit (Read-Only Ports)

```
  2134h - MPYL    - PPU1 Signed Multiply Result   (lower 8bit)     (01h)
  2135h - MPYM    - PPU1 Signed Multiply Result   (middle 8bit)    (00h)
  2136h - MPYH    - PPU1 Signed Multiply Result   (upper 8bit)     (00h)
  2137h - SLHV    - PPU1 Latch H/V-Counter by Software (Read=Strobe)
  2138h - RDOAM   - PPU1 OAM Data Read            (read-twice)
  2139h - RDVRAML - PPU1 VRAM Data Read           (lower 8bits)
  213Ah - RDVRAMH - PPU1 VRAM Data Read           (upper 8bits)
  213Bh - RDCGRAM - PPU2 CGRAM Data Read (Palette)(read-twice)
  213Ch - OPHCT   - PPU2 Horizontal Counter Latch (read-twice)     (01FFh)
  213Dh - OPVCT   - PPU2 Vertical Counter Latch   (read-twice)     (01FFh)
  213Eh - STAT77  - PPU1 Status and PPU1 Version Number
  213Fh - STAT78  - PPU2 Status and PPU2 Version Number            Bit7=0
```

APU Audio Processing Unit (R/W)

```
  2140h - APUI00  - Main CPU to Sound CPU Communication Port 0        (00h/00h)
  2141h - APUI01  - Main CPU to Sound CPU Communication Port 1        (00h/00h)
  2142h - APUI02  - Main CPU to Sound CPU Communication Port 2        (00h/00h)
  2143h - APUI03  - Main CPU to Sound CPU Communication Port 3        (00h/00h)
  2144h..217Fh    - APU Ports 2140-2143h mirrored to 2144h..217Fh
```

WRAM Access

```
  2180h - WMDATA  - WRAM Data Read/Write       (R/W)
  2181h - WMADDL  - WRAM Address (lower 8bit)  (W)                      00h
  2182h - WMADDM  - WRAM Address (middle 8bit) (W)                      00h
  2183h - WMADDH  - WRAM Address (upper 1bit)  (W)                      00h
  2184h..21FFh    - Unused region (open bus) / Expansion (B-Bus)        -
  2200h..3FFFh    - Unused region (open bus) / Expansion (A-Bus)        -
```

CPU On-Chip I/O Ports

```
  4000h..4015h        - Unused region (open bus)      ;\These ports have  -
  4016h/Write - JOYWR - Joypad Output (W)             ; long waitstates   00h
  4016h/Read  - JOYA  - Joypad Input Register A (R)   ; (1.78MHz cycles)  -
  4017h/Read  - JOYB  - Joypad Input Register B (R)   ; (all other ports  -
  4018h..41FFh        - Unused region (open bus)      ;/are 3.5MHz fast)  -
```

CPU On-Chip I/O Ports (Write-only) (Read=open bus)

```
  4200h - NMITIMEN- Interrupt Enable and Joypad Request                  00h
  4201h - WRIO    - Joypad Programmable I/O Port (Open-Collector Output) FFh
  4202h - WRMPYA  - Set unsigned 8bit Multiplicand                       (FFh)
  4203h - WRMPYB  - Set unsigned 8bit Multiplier and Start Multiplication (FFh)
  4204h - WRDIVL  - Set unsigned 16bit Dividend (lower 8bit)             (FFh)
  4205h - WRDIVH  - Set unsigned 16bit Dividend (upper 8bit)             (FFh)
  4206h - WRDIVB  - Set unsigned 8bit Divisor and Start Division         (FFh)
  4207h - HTIMEL  - H-Count Timer Setting (lower 8bits)                  (FFh)
  4208h - HTIMEH  - H-Count Timer Setting (upper 1bit)                   (01h)
  4209h - VTIMEL  - V-Count Timer Setting (lower 8bits)                  (FFh)
  420Ah - VTIMEH  - V-Count Timer Setting (upper 1bit)                   (01h)
  420Bh - MDMAEN  - Select General Purpose DMA Channel(s) and Start Transfer 0
  420Ch - HDMAEN  - Select H-Blank DMA (H-DMA) Channel(s)                  0
  420Dh - MEMSEL  - Memory-2 Waitstate Control                             0
  420Eh..420Fh    - Unused region (open bus)                               -
```

CPU On-Chip I/O Ports (Read-only)

```
  4210h - RDNMI   - V-Blank NMI Flag and CPU Version Number (Read/Ack)     0xh
  4211h - TIMEUP  - H/V-Timer IRQ Flag (Read/Ack)                          00h
  4212h - HVBJOY  - H/V-Blank flag and Joypad Busy flag (R)                (?)
  4213h - RDIO    - Joypad Programmable I/O Port (Input)                   -
  4214h - RDDIVL  - Unsigned Division Result (Quotient) (lower 8bit)       (0)
  4215h - RDDIVH  - Unsigned Division Result (Quotient) (upper 8bit)       (0)
  4216h - RDMPYL  - Unsigned Division Remainder / Multiply Product (lower 8bit)
  4217h - RDMPYH  - Unsigned Division Remainder / Multiply Product (upper 8bit)
  4218h - JOY1L   - Joypad 1 (gameport 1, pin 4) (lower 8bit)              00h
  4219h - JOY1H   - Joypad 1 (gameport 1, pin 4) (upper 8bit)              00h
  421Ah - JOY2L   - Joypad 2 (gameport 2, pin 4) (lower 8bit)              00h
  421Bh - JOY2H   - Joypad 2 (gameport 2, pin 4) (upper 8bit)              00h
  421Ch - JOY3L   - Joypad 3 (gameport 1, pin 5) (lower 8bit)              00h
  421Dh - JOY3H   - Joypad 3 (gameport 1, pin 5) (upper 8bit)              00h
  421Eh - JOY4L   - Joypad 4 (gameport 2, pin 5) (lower 8bit)              00h
  421Fh - JOY4H   - Joypad 4 (gameport 2, pin 5) (upper 8bit)              00h
  4220h..42FFh    - Unused region (open bus)                               -
```

CPU DMA, For below ports, x = Channel number 0..7 (R/W)

```
  (additional DMA control registers are 420Bh and 420Ch, see above)
  43x0h - DMAPx   - DMA/HDMA Parameters                                  (FFh)
  43x1h - BBADx   - DMA/HDMA I/O-Bus Address (PPU-Bus aka B-Bus)         (FFh)
  43x2h - A1TxL   - HDMA Table Start Address (low)  / DMA Curr Addr (low) (FFh)
  43x3h - A1TxH   - HDMA Table Start Address (high) / DMA Curr Addr (high)(FFh)
  43x4h - A1Bx    - HDMA Table Start Address (bank) / DMA Curr Addr (bank)(xxh)
  43x5h - DASxL   - Indirect HDMA Address (low)  / DMA Byte-Counter (low) (FFh)
  43x6h - DASxH   - Indirect HDMA Address (high) / DMA Byte-Counter (high)(FFh)
  43x7h - DASBx   - Indirect HDMA Address (bank)                         (FFh)
  43x8h - A2AxL   - HDMA Table Current Address (low)                     (FFh)
  43x9h - A2AxH   - HDMA Table Current Address (high)                    (FFh)
  43xAh - NTRLx   - HDMA Line-Counter (from current Table entry)         (FFh)
  43xBh - UNUSEDx - Unused byte (read/write-able)                        (FFh)
  43xCh+  -         Unused region (open bus)                              -
  43xFh - MIRRx   - Mirror of 43xBh (R/W)                                (FFh)
  4380h..5FFFh    - Unused region (open bus)                             -
```

Further Memory

```
  6000h..7FFFh    - Expansion (eg. Battery Backed RAM, in HiROM cartridges)
  8000h..FFFFh    - Cartridge ROM
```

Note: The right column shows the initial value on Reset, values in brackets are left unchanged upon Reset (but do contain the specified value upon initial Power-up).

Audio Registers (controlled by the SPC700 CPU, not by the Main CPU)

SNES APU Memory and I/O Map

Expansion Overview

```
  0000h-003Fh  Cheat Device: Pro Action Replay 1-3: I/O Ports; overlapping WRAM
  2184h-218Fh  Copier: Super UFO models Pro-7 and Pro-8
  2188h-2199h  Satellaview Receiver Unit (connected to Expansion Port)
  21C0h-21C3h  More or less "used" by Nintendo's "SNES Test" cartridge
  21C0h-21DFh  Exertainment (exercise bicycle) (connected to Expansion Port)
  21FCh-21FFh  Nocash Debug Extension (char_out and 21mhz_timer in no$sns emu)
  2200h-230Eh  SA-1 (programmable 65C816 CPU) I/O Ports
  2400h-2401h  Nintendo Power (flashcard)
  2800h-2801h  S-RTC Real Time Clock I/O Ports
  2800h-2810h  Copier: FDC I/O Ports CCL (Supercom Partner & Pro Fighter)
  2C00h-2FFFh  Cheat Device: X-Terminator 2: SRAM (32Kbytes, in banks 00h-1Fh)
  3000h-32FFh  GSU-n (programmable RISC CPU) I/O Ports
  3000h-37FFh  SA-1 (programmable 65C816 CPU) on-chip I-RAM
  3800h-3804h  ST018 (Seta) (pre-programmed ARM CPU) (maybe also FF40h..FF63h)
  4100h        Nintendo Super System (NSS) DIP-Switches (on game cartridge)
  4800h-4807h  S-DD1 Data Decompression chip
  4800h-4842h  SPC7110 Data Decompression chip (optionally with RTC-4513)
  5000h-5FFFh  Satellaview MCC mem ctrl (sixteen 1bit I/O ports banks 00h-0Fh)
  5000h-5FFFh  Satellaview 32Kbyte SRAM (eight 4Kbyte-chunk in banks 10h-17h)
  58xxh        Copier: Venus (Multi Game Hunter)
  5Fxxh        Copier: Gamars Super Disk
```

```
6000h-7FFFh  SRAM Battery Backed Static RAM (in HiROM cartridges) (bank 3xh)
6000h-7FFFh  SGB Super Gameboy I/O Ports
6000h-7FFFh  DSP-n on HiROM boards (pre-programmed NEC uPD77C25 CPU)
7F40h-7FAFh  CX4 I/O Ports (with 3K SRAM at 6000h..6BFFh)
7FF0h-7FF7h  OBC1 OBJ Controller I/O Ports (with 8K SRAM at 6000h..7FFFh)
8000h-FFFFh  Cartridge ROM (including header/exception vectors at FFxxh)
8000h        Pirate X-in-1 Multicart 32K-ROM bank (mapping 20 small games)
8000h-FFFFh  Copier: Various models map ROM, I/O, SRAM, DRAM in ROM area
8000h-8101h  Cheat Device: Game Genie I/O Ports (in ROM banks 00h and FFh)
A000h-A007h  Cheat Device: Pro Action Replay 2: I/O Control (in HiROM area)
FFE0h-FFFFh  Exception Vectors (variable in SA-1 and CX4) (fixed in GSU)
FFE8h-FFEAh  Cheat Device: X-Terminator 1-2: I/O Ports (in LoROM area)
FFF0h-FFF3h  Tri-Star/Super 8: NES Joypad 1-2, BIOS-disable, A/V-select
xF8000h-3FFFFFh  DSP-n on LoROM boards (pre-programmed NEC uPD77C25 CPU)
600000h-6F7FFFh  DSP-n on 2Mbyte-LoROM boards (planned/prototype only)
600000h-67FFFFh  ST010/ST011 Command/Status/Parameters I/O Ports
680000h-6FFFFFh  ST010/ST011 On-chip Battery-backed RAM
700000h-7xxxxxh  SRAM Battery Backed Static RAM (in LoROM cartridges)
808000h-BFFFFFh  Bootleg Copy-Protection I/O Ports
C00000h-FFFFFFh  Satellaview FLASH Cartridges (Detect/Write/Erase commands)
C00000h-Cn7FFFh  JRA PAT Backup FLASH Memory  (Detect/Write/Erase commands)
C08000h-FFFFFFh  Satellaview-like FLASH Data Packs in LoROM Cartridges
E00000h-FFFFFFh  Satellaview-like FLASH Data Packs in HiROM Cartridges
E00000h-E0FFFFh  X-Band Modem 64K SRAM (in two 32Kx8 chips)
FBC000h-FBC1BFh  X-Band Modem I/O Ports (mainly in this area)
FFFF00h-FFFFFFh  Pirate X-in-1 multicart mapper I/O port
```

SNES Memory

SNES Memory Map

SNES Memory Control

Work RAM (WRAM)
Work RAM is mapped directly to the CPU bus, and can be additionally accessed indirectly via I/O ports (mainly for DMA transfer purposes).
SNES Memory Work RAM Access

Video Memory (OAM/VRAM/CGRAM)
All video memory can be accessed only during V-Blank, or Forced Blank.
Video memory isn't mapped to the CPU bus, and can be accessed only via I/O ports (for bigger transfers, this would be usually done via DMA).
SNES Memory OAM Access (Sprite Attributes)
SNES Memory VRAM Access (Tile and BG Map)
SNES Memory CGRAM Access (Palette Memory)
Access during H-Blank doesn't seem to work too well - it is possible to change palette entries during H-Blank, but seems to work only during a few clock cycles, not during the full H-blank period.

Sound RAM
Sound RAM is mapped to a separate SPC700 CPU, not to the Main CPU. Accordingly, Sound RAM cannot be directly accessed by the Main CPU (nor by DMA). Instead, data transfers must be done by using some CPU-to-CPU software communication protocol. Upon Reset, this done by a Boot-ROM on the SPC700 side. For details, see:
SNES APU Main CPU Communication Port

DMA Transfers
DMA can be used to quickly transfer memory blocks to/from most memory locations (except Sound RAM isn't accessible via DMA, and WRAM-to-WRAM transfers don't work).
SNES DMA Transfers

SNES Memory Map

The SNES uses a 24bit address bus (000000h-FFFFFFh). These 24bit addresses are often divided into 8bit bank numbers (00h-FFh) plus 16bit offset (0000h-FFFFh). Some of these banks are broken into two 32Kbyte halves (0000h-7FFFh=System Area, 8000h-FFFFh=Cartridge ROM). Moreover, memory is divided into WS1 and WS2 areas, which can be configured to have different waitstates.

Overall Memory Map

```
Bank    Offset       Content                                    Speed
00h-3Fh:0000h-7FFFh  System Area (8K WRAM, I/O Ports, Expansion)  see below
00h-3Fh:8000h-FFFFh  WS1 LoROM (max 2048 Kbytes) (64x32K)        3.58MHz
   (00h:FFE0h-FFFFh) CPU Exception Vectors (Reset,Irq,Nmi,etc.)   3.58MHz
40h-7Dh:0000h-FFFFh  WS1 HiROM (max 3968 Kbytes) (62x64K)        3.58MHz
7Eh-7Fh:0000h-FFFFh  WRAM (Work RAM, 128 Kbytes) (2x64K)         3.58MHz
80h-BFh:0000h-7FFFh  System Area (8K WRAM, I/O Ports, Expansion)  see below
80h-BFh:8000h-FFFFh  WS2 LoROM (max 2048 Kbytes) (64x32K)        max 2.68MHz
C0h-FFh:0000h-FFFFh  WS2 HiROM (max 4096 Kbytes) (64x64K)        max 2.68MHz
```

Internal memory regions are WRAM and memory mapped I/O ports.
External memory regions are LoROM, HiROM, and Expansion areas.
Additional memory (not mapped to CPU addresses) (accessible only via I/O):

```
OAM          (512+32 bytes) (256+16 words)
VRAM         (64 Kbytes)    (32 Kwords)
Palette      (512 bytes)    (256 words)
Sound RAM    (64 Kbytes)
Sound ROM    (64 bytes BIOS Boot ROM)
```

System Area (banks 00h-3Fh and 80h-BFh)

```
Offset       Content                                        Speed
0000h-1FFFh  Mirror of 7E0000h-7E1FFFh (first 8Kbyte of WRAM)  3.58MHz
2000h-20FFh  Unused                                         2.68MHz
2100h-21FFh  I/O Ports (B-Bus)                              2.68MHz
2200h-3FFFh  Unused                                         2.68MHz
4000h-41FFh  I/O Ports (manual joypad access)               1.78MHz
4200h-5FFFh  I/O Ports                                      2.68MHz
```

```
  6000h-7FFFh  Expansion                                3.58MHz
```

For details on the separate I/O ports (and Expansion stuff), see:
SNES I/O Map

Cartridge ROM Capacity
The 24bit address bus allows to address 16MB, but wide parts are occupied by WRAM and I/O mirrors, which leaves only around 11.9MB for cartridge ROM in WS1/WS2 LoROM/HiROM regions (or more when also using Expansion regions and gaps in I/O area). In most cartridges, WS1 and WS2 are mirrors of each other, and most games do use only the LoROM, or only the HiROM areas, resulting in following capacities:

```
  LoROM games --> max 2MByte ROM (banks 00h-3Fh, with mirror at 80h-BFh)
  HiROM games --> max 4MByte ROM (banks 40h-7Dh, with mirror at C0h-FFh)
```

There are several ways to overcome that limits: Some LoROM games map additional "LoROM" banks into HiROM area (BigLoROM), or into WS2 area (SpecialLoROM). Some HiROM games map additional HiROM banks into WS2 area (ExHiROM). And some cartridges do use bank switching (eg. SA-1, S-DD1, SPC7110, and X-in-1 multicarts).

32K LoROM (32K ROM banks with System Area in the same bank)
ROM is broken into non-continous 32K blocks. The advantage is that one can access ROM and System Area (I/O ports and WRAM) without needing to change the CPU's current DB and PB register settings.

HiROM (plain 64K ROM banks)
HiROM mapping provides continous ROM addresses, but doesn't include I/O and WRAM regions "inside" of the ROM-banks.
The upper halves of the 64K-HiROM banks are usually mirrored to the corresponding 32K-LoROM banks (that is important for mapping the Interrupt and Reset Vectors from 40FFE0h-40FFFFh to 00FFE0h-00FFFFh).

Battery-backed SRAM
Battery-backed SRAM is used for saving game positions in many games. SRAM size is usually 2Kbyte, 8Kbyte, or 32Kbyte (or more than 32Kbyte in a few games).
There are two basic SRAM mapping schemes, one for LoROM games, and one for HiROM games:

```
  HiROM ---> SRAM at 30h-3Fh,B0h-BFh:6000h-7FFFh    ;small 8K SRAM bank(s)
  LoROM ---> SRAM at 70h-7Dh,F0h-FFh:0000h-7FFFh    ;big 32K SRAM bank(s)
```

SRAM is usually also mirrored to both WS1 and WS2 areas. SRAM in bank 30h-3Fh is often also mirrored to 20h-2Fh (or 10h-1Fh in some cases). SRAM in bank 70h-7Dh is sometimes crippled to 70h-71h or 70h-77h, or extended to 60h-7Dh, and sometimes also mirrored to offset 8000h-FFFFh.

A-Bus and B-Bus
Aside from the 24bit address bus (A-Bus), the SNES is having a second 8bit address bus (B-bus), used to access certain I/O ports. Both address busses are sharing the same data bus, but each bus is having its own read and write signals.
The CPU can access the B-Bus at offset 2100h-21FFh within the System Area (ie. for CPU accesses, the B-Bus is simply a subset of the A-Bus).
The DMA controller can access both B-Bus and A-Bus at once (ie. it can output source & destination addresses simultaneously to the two busses, allowing it to "read-and-write" in a single step, instead of using separate "read-then-write" steps).

Bank Switching
Most SNES games are satisfied with the 24bit address space. Bank switching is used only in a few games with special chips:

```
  S-DD1, SA-1, and SPC7110 chips (with mappable 1MByte-banks)
  Satellaview FLASH carts (can enable/disable ROM, PSRAM, FLASH)
  Nintendo Power FLASH carts (can map FLASH and SRAM to desired address)
  Pirate X-in-1 multicart mappers (mappable offset in 256Kbyte units)
  Cheat devices (and X-Band modem) can map their BIOS and can patch ROM bytes
  Copiers can map internal BIOS/DRAM/SRAM and external Cartridge memory
  Hotel Boxes (eg. SFC-Box) can map multiple games/cartridges
```

And, at the APU side, one can enable/disable the 64-byte boot ROM.

SNES Memory Control

420Dh - MEMSEL - Memory-2 Waitstate Control (W)

```
  7-1   Not used
  0     Access Cycle for Memory-2 Area (0=2.68MHz, 1=3.58MHz) (0 on reset)
```

Memory-2 consists of address 8000h-FFFFh in bank 80h-BFh, and address 0000h-FFFFh in bank C0h-FFh. 3.58MHz high speed memory requires 120ns or faster ROMs/EPROMs. 2.68MHz memory requires 200ns or faster ROMs/EPROMs.

```
  2.684658 MHz = 21.47727 MHz / 8     ;same access time as WRAM
  3.579545 MHz = 21.47727 MHz / 6     ;faster access than WRAM
```

Programs that do use the 3.58MHz setting should also indicate this in the Cartridge header at [FFD5h].Bit4.
SNES Cartridge ROM Header

Forced Blank
Allows to access video memory at any time. See INIDISP Bit7, Port 2100h.

SNES Memory Work RAM Access

The SNES includes 128Kbytes of Work RAM, which can be accessed in several ways:

```
  The whole 128K are at 7E0000h-7FFFFFh.
  The first 8K are also mirrored to xx0000h-xx1FFFh (xx=00h..3Fh and 80h..BFh)
  Moreover (mainly for DMA purposes) it can be accessed via Port 218xh.
```

2180h - WMDATA - WRAM Data Read/Write (R/W)

```
  7-0   Work RAM Data
```

Simply reads or writes the byte at the address in [2181h-2183h], and does then increment the address by one.
Note: Despite of the fast access time on 2180h reads (faster than 7E0000h-7FFFFFh reads), there is no prefetching involved (reading 2180h always returns the currently addressed byte, even if one mixes it with writes to 2180h or to 7E0000h-7FFFFFh).

2181h - WMADDL - WRAM Address (lower 8bit) (W)
2182h - WMADDM - WRAM Address (middle 8bit) (W)
2183h - WMADDH - WRAM Address (upper 1bit) (W)
17bit Address (in Byte-steps) for addressing the 128Kbytes of WRAM via 2180h.

DMA Notes
WRAM-to-WRAM DMA isn't possible (neither in A-Bus to B-Bus direction, nor vice-versa). Externally, the separate address lines are there, but the WRAM chip is unable to process both at once.

Timing Notes
Note that WRAM is accessed at 2.6MHz. Meaning that all variables, stack, and program code in RAM will be slow. The SNES doesn't include any fast RAM. However, there are a few tricks to get "3.5MHz RAM":
* Sequential read from WRAM via [2180h] is 3.5MHz fast, and has auto-increment.
* DMA registers at 43x0h-43xBh provide 8x12 bytes of read/write-able "memory".
* External RAM could be mapped to 5000h-5FFFh (but usually it's at slow 6000h).
* External RAM could be mapped to C00000h-FFFFFFh (probably rarely done too).

Other Notes
The B-Bus feature with auto-increment is making it fairly easy to boot the SNES without any ROM/EPROM by simply writing program bytes to WRAM (and mirroring it to the Program and Reset vector to ROM area):
SNES Xboo Upload (WRAM Boot)
Interestingly, the WRAM-to-ROM Area mirroring seems to be stable even when ROM Area is set to 3.5MHz Access Time - so it's unclear why Nintendo has restricted normal WRAM Access to 2.6MHz - maybe some WRAM chips are slower than others, or maybe they become unstable at certain room temperatures.

SNES Memory OAM Access (Sprite Attributes)

2102h/2103h - OAMADDL/OAMADDH - OAM Address and Priority Rotation (W)
```
15    OAM Priority Rotation  (0=OBJ #0, 1=OBJ #N) (OBJ with highest priority)
9-14  Not used
7-1   OBJ Number #N (for OBJ Priority)   ;\bit7-1 are used for two purposes
8-0   OAM Address   (for OAM read/write) ;/
```
This register contains of a 9bit Reload value and a 10bit Address register (plus the priority flag). Writing to 2102h or 2103h does change the lower 8bit or upper 1bit of the Reload value, and does additionally copy the (whole) 9bit Reload value to the 10bit Address register (with address Bit0=0 so next access will be an even address).
Caution: During rendering, the PPU is destroying the Address register (using it internally for whatever purposes), after rendering (at begin of Vblank, ie. at begin of line 225/240, but only if not in Forced Blank mode) it reinitializes the Address from the Reload value; the same reload occurs also when deactivating forced blank anytime during the first scanline of vblank (ie. during line 225/240).

2104h - OAMDATA - OAM Data Write (W)
2138h - RDOAM - OAM Data Read (R)
```
1st Access: Lower 8bit (even address)
2nd Access: Upper 8bit (odd address)
```
Reads and Writes to EVEN and ODD byte-addresses work as follows:
```
Write to EVEN address      -->  set OAM_Lsb = Data    ;memorize value
Write to ODD address<200h  -->  set WORD[addr-1] = Data*256 + OAM_Lsb
Write to ANY address>1FFh  -->  set BYTE[addr] = Data
Read from ANY address      -->  return BYTE[addr]
```
The address is automatically incremented after every read or write access.
OAM Size is 220h bytes (addresses 220h..3FFh are mirrors of 200h..21Fh).

OAM Content
SNES PPU Sprites (OBJs)

SNES Memory VRAM Access (Tile and BG Map)

2115h - VMAIN - VRAM Address Increment Mode (W)
```
7     Increment VRAM Address after accessing High/Low byte (0=Low, 1=High)
6-4   Not used
3-2   Address Translation    (0..3 = 0bit/None, 8bit, 9bit, 10bit)
1-0   Address Increment Step (0..3 = Increment Word-Address by 1,32,128,128)
```
The address translation is intended for bitmap graphics (where one would have filled the BG Map by increasing Tile numbers), technically it does thrice left-rotate the lower 8, 9, or 10 bits of the Word-address:
```
Translation  Bitmap Type              Port [2116h/17h]    VRAM Word-Address
8bit rotate  4-color; 1 word/plane    aaaaaaaaYYYxxxxx --> aaaaaaaaxxxxxYYY
9bit rotate  16-color; 2 words/plane  aaaaaaaYYYxxxxxP --> aaaaaaaxxxxxPYYY
10bit rotate 256-color; 4 words/plane aaaaaaYYYxxxxxPP --> aaaaaaxxxxxPPYYY
```
Where "aaaaa" would be the normal address MSBs, "YYY" is the Y-index (within a 8x8 tile), "xxxxx" selects one of the 32 tiles per line, "PP" is the bit-plane index (for BGs with more than one Word per plane). For the intended result (writing rows of 256 pixels) the Translation should be combined with Increment Step=1.
For Mode 7 bitmaps one could eventually combine step 32/128 with 8bit/10bit rotate:
```
8bit-rotate/step32   aaaaaaaaXXXxxYYY --> aaaaaaaaxxYYYXXX
10bit-rotate/step128 aaaaaaXXXxxxxYYY --> aaaaaaxxxxYYYXXX
```
Though the SNES can't access enought VRAM for fullscreen Mode 7 bitmaps.
Step 32 (without translation) is useful for updating BG Map columns (eg. after horizontal scrolling).

2116h - VMADDL - VRAM Address (lower 8bit) (W)
2117h - VMADDH - VRAM Address (upper 8bit) (W)
VRAM Address for reading/writing. This is a WORD address (2-byte steps), the PPU could theoretically address up to 64K-words (128K-bytes), in practice, only 32K-words (64K-bytes) are installed in SNES consoles (VRAM address bit15 is not connected, so addresses 8000h-FFFFh are mirrors of 0-7FFFh).
After reading/writing VRAM Data, the Word-address can be automatically incremented by 1,32,128 (depending on the Increment Mode in Port 2115h) (Note: the Address Translation feature is applied only "temporarily" upon memory accesses, it doesn't affect the value in Port 2116h-17h).
Writing to 2116h/2117h does prefetch 16bit data from the new address (for later reading).

2118h - VMDATAL - VRAM Data Write (lower 8bit) (W)
2119h - VMDATAH - VRAM Data Write (upper 8bit) (W)
Writing to 2118h or 2119h does simply modify the LSB or MSB of the currently addressed VRAM word (with optional Address Translation applied). Depending on the Increment Mode the address does (or doesn't) get automatically incremented after the write.

2139h - RDVRAML - VRAM Data Read (lower 8bit) (R)
213Ah - RDVRAMH - VRAM Data Read (upper 8bit) (R)
Reading from these registers returns the LSB or MSB of an internal 16bit prefetch register. Depending on the Increment Mode the address does (or doesn't) get automatically incremented after the read.
The prefetch register is filled with data from the currently addressed VRAM word (with optional Address Translation applied) upon two situations:
```
Prefetch occurs AFTER changing the VRAM address (by writing 2116h/17h).
```

```
Prefetch occurs BEFORE incrementing the VRAM address (by reading 2139h/3Ah).
```
The "Prefetch BEFORE Increment" effect is some kind of a hardware glitch (Prefetch AFTER Increment would be more useful). Increment/Prefetch in detail:
```
1st  Send a byte from OLD prefetch value to the CPU        ;-this always
2nd  Load NEW value from OLD address into prefetch register;\these only if
3rd  Increment address so it becomes the NEW address       ;/increment occurs
```
Increments caused by writes to 2118h/19h don't do any prefetching (the prefetch register is left totally unchanged by writes).
In practice: After changing the VRAM address (via 2116h/17h), the first byte/word will be received twice, further values are received from properly increasing addresses (as a workaround: issue a dummy-read that ignores the 1st or 2nd value).

VRAM Content
SNES PPU Video Memory (VRAM)

SNES Memory CGRAM Access (Palette Memory)

2121h - CGADD - Palette CGRAM Address (Color Generator Memory) (W)
Color index (0..255). This is a WORD-address (2-byte steps), allowing to access 256 words (512 bytes). Writing to this register resets the 1st/2nd access flipflop (for 2122h/213Bh) to 1st access.

2122h - CGDATA - Palette CGRAM Data Write (W)
213Bh - RDCGRAM - Palette CGRAM Data Read (R)
```
1st Access: Lower 8 bits (even address)
2nd Access: Upper 7 bits (odd address) (upper 1bit = PPU2 open bus)
```
Reads and Writes to EVEN and ODD byte-addresses work as follows:
```
Write to EVEN address  -->  set Cgram_Lsb = Data    ;memorize value
Write to ODD address   -->  set WORD[addr-1] = Data*256 + Cgram_Lsb
Read from ANY address  -->  return BYTE[addr]
```
The address is automatically incremented after every read or write access.

CGRAM Content (and CGRAM-less Direct Color mode)
SNES PPU Color Palette Memory (CGRAM) and Direct Colors

SNES DMA Transfers

The SNES includes eight DMA channels, which can be used for H-DMA or GP-DMA.
SNES DMA and HDMA Start/Enable Registers
SNES DMA and HDMA Channel 0..7 Registers
SNES DMA and HDMA Notes

H-DMA (H-Blank DMA)
H-DMA transfers are automatically invoked on H-Blank, each H-DMA is limited to a single unit (max 4 bytes) per scanline. This is commonly used to manipulate PPU I/O ports (eg. to change scroll offsets). Related registers can found here:
SNES I/O Map
SNES Picture Processing Unit (PPU)

GP-DMA (General Purpose DMA)
GP-DMA can manually invoked by software, allowing to transfer larger amounts of data (max 10000h bytes). This is commonly used to transfer WRAM or ROM (on A-Bus side) to/from WRAM, OAM, VRAM, CGRAM (on B-Bus side). Related registers are:
SNES Memory Work RAM Access
SNES Memory OAM Access (Sprite Attributes)
SNES Memory VRAM Access (Tile and BG Map)
SNES Memory CGRAM Access (Palette Memory)

SNES DMA and HDMA Start/Enable Registers

```
DMA and HDMA Transfer order is Channel 0 first... Channel 7 last
HDMA has higher prio than DMA
HDMA is running even during Forced Blank.
```

420Bh - MDMAEN - Select General Purpose DMA Channel(s) and Start Transfer (W)
```
7-0   General Purpose DMA Channel 7-0 Enable (0=Disable, 1=Enable)
```
When writing a non-zero value to this register, general purpose DMA will be started immediately (after a few clk cycles). The CPU is paused during the transfer. The transfer can be interrupted by H-DMA transfers. If more than 1 bit is set in MDMAEN, then the separate transfers will be executed in order channel 0=first through 7=last. The MDMAEN bits are cleared automatically at transfer completion.
Do not use channels for GP-DMA which are activated as H-DMA in HDMAEN.

420Ch - HDMAEN - Select H-Blank DMA (H-DMA) Channel(s) (W)
```
7-0   H-DMA Channel 7-0 Enable (0=Disable, 1=Enable)
```
...

SNES DMA and HDMA Channel 0..7 Registers

For below ports, x = Channel number (0-7)

43x0h - DMAPx - DMA/HDMA Parameters (R/W)
```
7     Transfer Direction (0=A:CPU to B:I/O, 1=B:I/O to A:CPU)
6     Addressing Mode    (0=Direct Table, 1=Indirect Table)    (HDMA only)
5     Not used (R/W) (unused and unchanged by all DMA and HDMA)
4-3   A-BUS Address Step  (0=Increment, 2=Decrement, 1/3=Fixed) (DMA only)
2-0   Transfer Unit Select (0-4=see below, 5-7=Reserved)
```
DMA Transfer Unit Selection:
```
Mode  Bytes             B-Bus 21xxh Address   ;Usage Examples...
```

```
  0  =  Transfer 1 byte    xx                  ;eg. for WRAM (port 2180h)
  1  =  Transfer 2 bytes   xx, xx+1            ;eg. for VRAM (port 2118h/19h)
  2  =  Transfer 2 bytes   xx, xx              ;eg. for OAM or CGRAM
  3  =  Transfer 4 bytes   xx, xx,   xx+1, xx+1  ;eg. for BGnxOFS, M7x
  4  =  Transfer 4 bytes   xx, xx+1, xx+2, xx+3  ;eg. for BGnSC, Window, APU..
  5  =  Transfer 4 bytes   xx, xx+1, xx,   xx+1  ;whatever purpose, VRAM maybe
  6  =  Transfer 2 bytes   xx, xx              ;same as mode 2
  7  =  Transfer 4 bytes   xx, xx,   xx+1, xx+1  ;same as mode 3
```

A HDMA transfers ONE unit per scanline (=max 4 bytes). General Purpose DMA has a 16bit length counter, allowing to transfer up to 10000h bytes (ie. not 10000h units).

43x1h - BBADx - DMA/HDMA I/O-Bus Address (PPU-Bus aka B-Bus) (R/W)
For both DMA and HDMA:

```
  7-0   B-Bus Address (selects an I/O Port which is mapped to 2100h-21FFh)
```

For normal DMA this should be usually 04h=OAM, 18h=VRAM, 22h=CGRAM, or 80h=WRAM. For HDMA it should be usually some PPU register (eg. for changing scroll offsets midframe).

43x2h - A1TxL - HDMA Table Start Address (low) / DMA Current Addr (low) (R/W)
43x3h - A1TxH - HDMA Table Start Address (hi) / DMA Current Addr (hi) (R/W)
43x4h - A1Bx - HDMA Table Start Address (bank) / DMA Current Addr (bank) (R/W)
For normal DMA:

```
  23-16  CPU-Bus Data Address Bank (constant, not incremented/decremented)
  15-0   CPU-Bus Data Address (incremented/decremented/fixed, as selected)
```

For HDMA:

```
  23-16  CPU-Bus Table Address Bank (constant, bank number for 43x8h/43x9h)
  15-0   CPU-Bus Table Address      (constant, reload value for 43x8h/43x9h)
```

43x5h - DASxL - Indirect HDMA Address (low) / DMA Byte-Counter (low) (R/W)
43x6h - DASxH - Indirect HDMA Address (hi) / DMA Byte-Counter (hi) (R/W)
43x7h - DASBx - Indirect HDMA Address (bank) (R/W)
For normal DMA:

```
  23-16  Not used
  15-0   Number of bytes to be transferred (1..FFFFh=1..FFFFh, or 0=10000h)
  (This is really a byte-counter; with a 4-byte "Transfer Unit", len=5 would
  transfer one whole Unit, plus the first byte of the second Unit.)
  (The 16bit value is decremented during transfer, and contains 0000h on end.)
```

For HDMA in direct mode:

```
  23-0   Not used     (in this mode, the Data is read directly from the Table)
```

For HDMA in indirect mode:

```
  23-16  Current CPU-Bus Data Address Bank   (this must be set by software)
  16-0   Current CPU-Bus Data Address (automatically loaded from the Table)
```

43x8h - A2AxL - HDMA Table Current Address (low) (R/W)
43x9h - A2AxH - HDMA Table Current Address (high) (R/W)
For normal DMA:

```
  15-0  Not used
```

For HDMA:

```
  -     Current Table Address Bank (taken from 43x4h)
  15-0  Current Table Address (reloaded from 43x2h/43x3h) (incrementing)
```

43xAh - NTRLx - HDMA Line-Counter (from current Table entry) (R/W)
For normal DMA:

```
  7-0   Not used
```

For HDMA:

```
  7     Repeat-flag                       ;\(loaded from Table, and then
  6-0   Number of lines to be transferred   ;/decremented per scanline)
```

43xBh - UNUSEDx - Unused Byte (R/W)

```
  7-0   Not used (read/write-able)
```

Can be used as a fast RAM location (but NOT as a fixed DMA source address for memfill). Storing any value in this register seems to have no effect on the transfer (and the value is left intact, not modified by DMA nor direct nor indirect HDMAs).

43xCh..43xEh - Unused region (open bus)
Unused. Reading returns garbage (open bus), writing seems to have no effect, even when trying to "disturb" HDMAs.

43xFh - MIRRx - Read/Write-able mirror of 43xBh (R/W)
Mirror of 43xBh.

HDMA Table Formats (in Direct and Indirect Mode)
In Direct Mode, the table consists of entries in following format:

```
  1 byte   Repeat-flag & line count
  N bytes  Data (where N=unit size, if repeat=1: multiplied by line count)
```

In Indirect Mode, the table consists of entries in following format:

```
  1 byte   Repeat-flag & line count
  2 bytes  16bit pointer to N bytes of Data (where N = as for Direct HDMA)
```

In either mode: The "repeat-flag & line count" bytes can be:

```
  00h       Terminate this HDMA channel (until it restarts in next frame)
  01h..80h  Transfer 1 unit in 1 line, then pause for next "X-01h" lines
  81h..FFh  Transfer X-80h units in X-80h lines ("repeat mode")
```

The "count" and "pointer" values are always READ from the table. The "data" values are READ or WRITTEN depending on the transfer direction. The transfer step is always INCREMENTING for HDMA (for both the table itself, as well as for any indirectly addressed data blocks).

SNES DMA and HDMA Notes

Starting HDMA midframe
Activating HDMA (via port 420Ch) should be usually done only during VBlank (so that the hardware can reload the HDMA registers automatically at end of Vblank).
Otherwise, when starting HDMA midframe (outside of Vblank), then one must manually reload the HDMA registers. For example, during Vblank, init HDMA registers as so:

```
  420Ch=00h ;stop all HDMA channels
```

```
43x0h=02h ;transfer two bytes to [bbus+0], and [bbus+0]
43x1h=88h ;dummy bbus destination address (unused port 2188h)
42x4h=abus.src.bank       ;with <abus.src> pointing to "02h,77h,55h,00h"
42x8h=abus.src.offs.lo    ;ie. repeat/pause 2 scanlines (02h), transfer
42x9h=abus.src.off.hi     ;one data unit (77h,55h), and after the pause,
42xAh=01h ;remain count   ;finish the transfer (00h).
```

The HDMA starting point seems to depend on whether any HDMA channels were already active in the current frame. The two scenarios (each with above example values) are:

Case 1 - (420Ch was still zero) - First HDMA in current frame

Start one (or more) HDMA channel(s) somewhere midframe (eg. in line 128), and watch the src/remain values in 43x8h..43xAh. This will behave as expected (src increases, and remain decreases from 02h downto 00h).

Case 2 - (420Ch was already nonzero) - Further HDMA in current frame

Start another HDMA channel (some scanlines later, after the above transfer). This will behave differently: It's decreasing remain count in 43xAh from 55h downwards, ie. the HDMA does apparently start with "do_transfer=1" (for whatever reason), causing it to transfer 02h,77h as data, and then fetch 55h as repeat count for next scanlines.

Note: One game does start HDMAs midframe is Super Ghouls 'N Ghosts.

[XXX Case 2 may need more research, and isn't yet accurately emulated in no$sns]

External DMA

The SNES Cartridge Slot doesn't have any special DRQ/DACK pins for DMA handshaking purposes; DMA from cartridge memory is just implemented as normal memory access (so the cartridge must respond to DMAs within normal memory access time; which is fixed 8 master cycles per byte for DMA).

Unknown if the data decompression chips (S-DD1 and SPC7110) are implemented the same way, or if they do use any "hidden" DMA handshaking mechanisms (they might be too slow to supply bytes within 8 master cycles, and at least one of them seems to decode DMA channel numbers; possibly by sensing writes to 420Bh?).

SNES Picture Processing Unit (PPU)

SNES PPU Control
SNES PPU BG Control
SNES PPU Rotation/Scaling
SNES PPU Window
SNES PPU Color-Math
SNES PPU Timers and Status
SNES PPU Interrupts
SNES PPU Resolution
SNES PPU Offset-Per-Tile Mode

Video Memory (OAM/VRAM/CGRAM)

SNES PPU Sprites (OBJs)
SNES PPU Video Memory (VRAM)
SNES PPU Color Palette Memory (CGRAM) and Direct Colors

All video memory can be accessed only during V-Blank, or Forced Blank.

Video memory isn't mapped to the CPU bus, and be accessed only via I/O ports.

SNES Memory OAM Access (Sprite Attributes)
SNES Memory VRAM Access (Tile and BG Map)
SNES Memory CGRAM Access (Palette Memory)

The above OAM/VRAM/CGRAM I/O ports are usually accessed via DMA,

SNES DMA Transfers

Pinouts

SNES Audio/Video Connector Pinouts
SNES Pinouts PPU Chips

Background Priority Chart

Mode0	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
-	BG3.1a	-	-	-	-	-	-
OBJ.3	OBJ.3	OBJ.3	OBJ.3	OBJ.3	OBJ.3	OBJ.3	OBJ.3
BG1.1	BG1.1	BG1.1	BG1.1	BG1.1	BG1.1	BG1.1	-
BG2.1	BG2.1	-	-	-	-	-	-
OBJ.2	OBJ.2	OBJ.2	OBJ.2	OBJ.2	OBJ.2	OBJ.2	OBJ.2
BG1.0	BG1.0	BG2.1	BG2.1	BG2.1	BG2.1	-	BG2.1p
BG2.0	BG2.0	-	-	-	-	-	-
OBJ.1	OBJ.1	OBJ.1	OBJ.1	OBJ.1	OBJ.1	OBJ.1	OBJ.1
BG3.1	BG3.1b	BG1.0	BG1.0	BG1.0	BG1.0	BG1.0	BG1
BG4.1	-	-	-	-	-	-	-
OBJ.0	OBJ.0	OBJ.0	OBJ.0	OBJ.0	OBJ.0	OBJ.0	OBJ.0
BG3.0	BG3.0a	BG2.0	BG2.0	BG2.0	BG2.0	-	BG2.0p
BG4.0	BG3.0b	-	-	-	-	-	-
Backdrop	Backdrop	Backdrop	Backdrop	Backdrop	Backdrop	Backdrop	Backdrop

Whereas,

```
.N     per-tile priority setting (in BG Map and OAM entries)
.Np    per-pixel priority setting (for 128-color BG2 in Mode7)
.Na/b  per-screen priority bit (in port 2105h) (plus .N as usually)
```

SNES PPU Control

2100h - INIDISP - Display Control 1 (W)

```
7     Forced Blanking (0=Normal, 1=Screen Black)
6-4   Not used
3-0   Master Brightness (0=Screen Black, or N=1..15: Brightness*(N+1)/16)
```

In Forced Blank, VRAM, OAM and CGRAM can be freely accessed (otherwise it's accessible only during Vblank). Even when in forced blank, the TV Set keeps receiving Vsync/Hsync signals (thus producing a stable black picture). And, the CPU keeps receiving Hblank/Vblank signals (so any enabled video NMIs, IRQs, HDMAs are kept generated).

```
Forced blank doesn't apply immediately... so one must wait whatever
(maybe a scanline) before VRAM can be freely accessed... or is it only
vice-versa: disabling forced blank doesn't apply immediately/shows garbage
pixels?
```

212Ch - TM - Main Screen Designation (W)
212Dh - TS - Sub Screen Designation (W)
```
7-5  Not used
4    OBJ (0=Disable, 1=Enable)
3    BG4 (0=Disable, 1=Enable)
2    BG3 (0=Disable, 1=Enable)
1    BG2 (0=Disable, 1=Enable)
0    BG1 (0=Disable, 1=Enable)
-    Backdrop (Always enabled)
```
Allows to enable/disable video layers. The Main screen is the "normal" display. The Sub screen is used only for Color Math and for 512-pixel Hires Mode.

2133h - SETINI - Display Control 2 (W)
```
7     External Synchronization (0=Normal, 1=Super Impose and etc.)
6     EXTBG Mode (Screen expand)
        ENABLE THE DATA SUPPLIED FROM THE EXTERNAL LSI.
        FOR THE SFX, ENABLE WHEN THE SCREEN WITH PRIORITY IS USED ON MODE-7.
5-4   Not used
3     Horizontal Pseudo 512 Mode (0=Disable, 1=Enable)
        (SHIFT SUBSCREEN HALF DOT TO THE LEFT)
2     BG V-Direction Display (0=224 Lines, 1=239 Lines) (for NTSC/PAL)
1     OBJ V-Direction Display (0=Low, 1=High Resolution/Smaller OBJs)
        IN THE INTERLACE MODE, SELECT EITHER OF 1-DOT PER LINE OR 1-DOT
        REPEATED EVERY 2-LINES. IF "1" IS WRITTEN, THE OBJ SEEMS REDUCED
        HALF VERTICALLY IN APPEARANCE.
0     V-Scanning         (0=Non Interlace, 1=Interlace) (See Port 2105h)
```

SNES PPU BG Control

2105h - BGMODE - BG Mode and BG Character Size (W)
```
7    BG4 Tile Size (0=8x8, 1=16x16)  ;\(BgMode0..4: variable 8x8 or 16x16)
6    BG3 Tile Size (0=8x8, 1=16x16)  ; (BgMode5: 8x8 acts as 16x8)
5    BG2 Tile Size (0=8x8, 1=16x16)  ; (BgMode6: fixed 16x8?)
4    BG1 Tile Size (0=8x8, 1=16x16)  ;/(BgMode7: fixed 8x8)
3    BG3 Priority in Mode 1 (0=Normal, 1=High)
2-0  BG Screen Mode (0..7 = see below)
```
The BG Screen Modes are:
```
Mode   BG1         BG2         BG3         BG4
0      4-color     4-color     4-color     4-color   ;Normal
1      16-color    16-color    4-color     -         ;Normal
2      16-color    16-color    (o.p.t)     -         ;Offset-per-tile
3      256-color   16-color    -           -         ;Normal
4      256-color   4-color     (o.p.t)     -         ;Offset-per-tile
5      16-color    4-color     -           -         ;512-pix-hires
6      16-color    -           (o.p.t)     -         ;512-pix plus Offs-p-t
7      256-color   EXTBG       -           -         ;Rotation/Scaling
```
Mode 7 supports rotation/scaling and EXTBG (but doesn't support hv-flip).
Mode 5/6 don't support screen addition/subtraction.
CG Direct Select is support on BG1 of Mode 3/4, and on BG1/BG2? of Mode 7.

2106h - MOSAIC - Mosaic Size and Mosaic Enable (W)
Allows to divide the BG layer into NxN pixel blocks, in each block, the hardware picks the upper-left pixel of each block, and fills the whole block by the color - thus effectively reducing the screen resolution.
```
7-4  Mosaic Size        (0=Smallest/1x1, 0Fh=Largest/16x16)
3    BG4 Mosaic Enable  (0=Off, 1=On)
2    BG3 Mosaic Enable  (0=Off, 1=On)
1    BG2 Mosaic Enable  (0=Off, 1=On)
0    BG1 Mosaic Enable  (0=Off, 1=On)
```
Horizontally, the first block is always located on the left edge of the TV screen. Vertically, the first block is located on the top of the TV screen.
When changing the mosaic size mid-frame, the hardware does first finish current block (using the old vertical size) before applying the new vertical size.
Technically, vertical mosaic is implemented as so: subtract the veritical index (within the current block) from the vertical scroll register (BGnVOFS).

2107h - BG1SC - BG1 Screen Base and Screen Size (W)
2108h - BG2SC - BG2 Screen Base and Screen Size (W)
2109h - BG3SC - BG3 Screen Base and Screen Size (W)
210Ah - BG4SC - BG4 Screen Base and Screen Size (W)
```
7-2  SC Base Address in VRAM (in 1K-word steps, aka 2K-byte steps)
1-0  SC Size (0=One-Screen, 1=V-Mirror, 2=H-Mirror, 3=Four-Screen)
                (0=32x32, 1=64x32, 2=32x64, 3=64x64 tiles)
             (0: SC0 SC0    1: SC0 SC1  2: SC0 SC0  3: SC0 SC1   )
             (   SC0 SC0       SC0 SC1     SC1 SC1     SC2 SC3   )
```
Specifies the BG Map addresses in VRAM. The "SCn" screens consists of 32x32 tiles each.
Ignored in Mode 7 (Base is always zero, size is always 128x128 tiles).

210Bh/210Ch - BG12NBA/BG34NBA - BG Character Data Area Designation (W)
```
15-12 BG4 Tile Base Address (in 4K-word steps)
11-8  BG3 Tile Base Address (in 4K-word steps)
7-4   BG2 Tile Base Address (in 4K-word steps)
3-0   BG1 Tile Base Address (in 4K-word steps)
```
Ignored in Mode 7 (Base is always zero).

210Dh - BG1HOFS - BG1 Horizontal Scroll (X) (W) and M7HOFS
210Eh - BG1VOFS - BG1 Vertical Scroll (Y) (W) and M7VOFS
210Fh - BG2HOFS - BG2 Horizontal Scroll (X) (W)
2110h - BG2VOFS - BG2 Vertical Scroll (Y) (W)
2111h - BG3HOFS - BG3 Horizontal Scroll (X) (W)
2112h - BG3VOFS - BG3 Vertical Scroll (Y) (W)
2113h - BG4HOFS - BG4 Horizontal Scroll (X) (W)
2114h - BG4VOFS - BG4 Vertical Scroll (Y) (W)
```
1st Write: Lower 8bit  ;\1st/2nd write mechanism uses "BG_old"
2nd Write: Upper 2bit  ;/
```
Note: Port 210Dh/210Eh are also used as M7HOFS/M7VOFS, these registers have a similar purpose, but internally they are separate registers: Writing to 210Dh

does BOTH update M7HOFS (via M7_old mechanism), and also updates BG1HOFS (via BG_old mechanism). In the same fashion, 210Eh updates both M7VOFS and BG1VOFS.

```
BGnHOFS = (Current<<8) | (Prev&~7) | ((Reg>>8)&7);
Prev = Current;
  or
BGnVOFS = (Current<<8) | Prev;
Prev = Current;
```

SNES PPU Rotation/Scaling

211Ah - M7SEL - Rotation/Scaling Mode Settings (W)

```
7-6   Screen Over (see below)
5-2   Not used
1     Screen V-Flip (0=Normal, 1=Flipped)     ;\flip 256x256 "screen"
0     Screen H-Flip (0=Normal, 1=Flipped)     ;/
```

Screen Over (when exceeding the 128x128 tile BG Map size):

```
0=Wrap within 128x128 tile area
1=Wrap within 128x128 tile area (same as 0)
2=Outside 128x128 tile area is Transparent
3=Outside 128x128 tile area is filled by Tile 00h
```

211Bh - M7A - Rotation/Scaling Parameter A (and Maths 16bit operand) (W)
211Ch - M7B - Rotation/Scaling Parameter B (and Maths 8bit operand) (W)
211Dh - M7C - Rotation/Scaling Parameter C (W)
211Eh - M7D - Rotation/Scaling Parameter D (W)

```
1st Write: Lower 8bit  ;\1st/2nd write mechanism uses "M7_old"
2nd Write: Upper 8bit  ;/
```

Signed 16bit values in 1/256 pixel units (1bit sign, 7bit integer, 8bit fraction).

210Dh - M7HOFS/BG1HOFS - BG1 Horizontal Scroll (X) (W)
210Eh - M7VOFS/BG1VOFS - BG1 Vertical Scroll (Y) (W)
211Fh - M7X - Rotation/Scaling Center Coordinate X (W)
2120h - M7Y - Rotation/Scaling Center Coordinate Y (W)

```
1st Write: Lower 8bit  ;\1st/2nd write mechanism uses "M7_old"
2nd Write: Upper 5bit  ;/
```

Signed 13bit values in pixel units (1bit sign, 12bit integer, 0bit fraction).

Formula

Formula for Rotation/Enlargement/Reduction in Matrix Form:

```
( VRAM.X )  =  ( M7A M7B )  *  ( SCREEN.X+M7HOFS-M7X )  +  ( M7X )
( VRAM.Y )     ( M7C M7D )     ( SCREEN.Y+M7VOFS-M7Y )     ( M7Y )
```

Parameters:

```
M7A=+COS(angle)*ScaleX, M7B=+SIN(angle)*ScaleX
M7C=-SIN(angle)*ScaleY, M7D=+COS(angle)*ScaleY
M7X,M7Y       = Center Coordinate
M7HOFS,M7VOFS = Scroll Offset
SCREEN.X = Display (Target) X-Coordinate: (0..255) XOR (xflip*FFh)
SCREEN.Y = Display (Target) Y-Coordinate: (1..224 or 1..239) XOR (yflip*FFh)
VRAM.X,Y = BG Map (Source) Coordinates (in 1/256 pixel units)
```

To calculate VRAM coordinates for any SCREEN coordinates:

```
IF xflip THEN SCREEN.X=((0..255) XOR FFh), ELSE SCREEN.X=(0..255)
IF yflip THEN SCREEN.Y=((1..224/239) XOR FFh), ELSE SCREEN.Y=(1..224/239)
ORG.X = (M7HOFS-M7X) AND NOT 1C00h, IF ORG.X<0 THEN ORG.X=ORG.X OR 1C00h
ORG.Y = (M7VOFS-M7Y) AND NOT 1C00h, IF ORG.Y<0 THEN ORG.Y=ORG.Y OR 1C00h
VRAM.X = ((M7A*ORG.X) AND NOT 3Fh) + ((M7B*ORG.Y) AND NOT 3Fh) + M7X*100h
VRAM.Y = ((M7C*ORG.X) AND NOT 3Fh) + ((M7D*ORG.Y) AND NOT 3Fh) + M7Y*100h
VRAM.X = VRAM.X + ((M7B*SCREEN.Y) AND NOT 3Fh) + (M7A*SCREEN.X)
VRAM.Y = VRAM.Y + ((M7D*SCREEN.Y) AND NOT 3Fh) + (M7C*SCREEN.X)
```

After calculating the left-most pixel of a scanline, the following pixels on that scanline can be also calculated by increasing VRAM coordinates as so:

```
IF xflip THEN VRAM.X=VRAM.X-M7A, ELSE VRAM.X=VRAM.X+M7A
IF xflip THEN VRAM.Y=VRAM.Y-M7C, ELSE VRAM.Y=VRAM.Y+M7C
(The result is same as on hardware, although the real hardware doesn't seem
to use that method, instead it seems to contain an excessively fast multiply
unit that recalculates (M7A*SCREEN.X) and (M7C*SCREEN.X) on every pixel.)
```

The VRAM coordinates are then: bit0-7=Fraction, bit8-10=pixel index (within a tile), bit11-17=map index (within BG map), bit18-and-up=nonzero when exceeding BG map size (do "screen over" handling).

M7A/M7B Port Notes

Port 211Bh/211Ch can be also used for general purpose math multiply:
SNES Maths Multiply/Divide
When in BG Mode 7, general purpose multiply works only during V-Blank and Forced-Blank. During drawing period at SCREEN.Y=0..224/239 (including SCREEN.Y=0, and during all 340 dots including H-Blank), MPYL/M/H receives two multiplication results per pixel (one per half-pixel):

```
MPY = M7A * ORG.X / 8                              ;at SCREEN.X=-3.0
MPY = M7D * ORG.Y / 8                              ;at SCREEN.X=-2.5
MPY = M7B * ORG.Y / 8                              ;at SCREEN.X=-2.0
MPY = M7C * ORG.X / 8                              ;at SCREEN.X=-1.5
MPY = M7B * ((SCREEN.Y-MOSAIC.Y) XOR (yflip*FFh))/ 8 ;at SCREEN.X=-1.0
MPY = M7D * ((SCREEN.Y-MOSAIC.Y) XOR (yflip*FFh))/ 8 ;at SCREEN.X=-0.5
MPY = M7A * ((SCREEN.X AND FFh) XOR (xflip*FFh)) / 8 ;at SCREEN.X=0.0..336.0
MPY = M7C * ((SCREEN.X AND FFh) XOR (xflip*FFh)) / 8 ;at SCREEN.X=0.5..336.5
MPY = M7A * (M7B/100h)                    ;during in V-Blank and Forced-Blank
```

Note: The "/8" suggests that the hardware strips the lower 3bit, however, before summing up the multiply results, it DOES strip the lower 6bit (hence the AND NOT 3Fh in the formula).

M7HOVS/M7VOFS Port Notes

Port 210Dh/210Eh are also used as BG1HOFS/BG1VOFS, these registers have a similar purpose, but internally they are separate registers: Writing to 210Dh does BOTH update M7HOFS (via M7_old mechanism), and also updates BG1HOFS (via BG_old mechanism). In the same fashion, 210Eh updates both M7VOFS and BG1VOFS.

M7xx - Write-twice mechanism for Mode 7

Writing a <new> byte to one of the write-twice M7'registers does:

```
M7_reg = new * 100h + M7_old
```

```
M7_old = new
```

M7_old is an internal 8bit register, shared by 210Dh-210Eh and 211Bh-2120h.

EXTBG

EXTBG is an "external" BG layer (replacing BG2???) enabled via SETINI.6. On the SNES, the 8bit external input is simply shortcut with one half of the PPUs 16bit data bus. So, when using EXTBG in BG Mode 0-6, one will just see garbage. However, in BG Mode 7, it's receiving the same 8bit value as the current BG1 pixel - but, unlike BG1, with bit7 treated as priority bit (and only lower 7bit used as BG2 pixel color).

SNES PPU Sprites (OBJs)

2101h - OBSEL - Object Size and Object Base (W)

```
7-5   OBJ Size Selection  (0-5, see below) (6-7=Reserved)
       Val Small  Large
       0 = 8x8    16x16    ;Caution:
       1 = 8x8    32x32    ;In 224-lines mode, OBJs with 64-pixel height
       2 = 8x8    64x64    ;may wrap from lower to upper screen border.
       3 = 16x16  32x32    ;In 239-lines mode, the same problem applies
       4 = 16x16  64x64    ;also for OBJs with 32-pixel height.
       5 = 32x32  64x64
       6 = 16x32  32x64 (undocumented)
       7 = 16x32  32x32 (undocumented)
      (Ie. a setting of 0 means Small OBJs=8x8, Large OBJs=16x16 pixels)
      (Whether an OBJ is "small" or "large" is selected by a bit in OAM)
4-3   Gap between OBJ 0FFh and 100h (0=None) (4K-word steps) (8K-byte steps)
2-0   Base Address for OBJ Tiles 000h..0FFh  (8K-word steps) (16K-byte steps)
```

Accessing OAM

SNES Memory OAM Access (Sprite Attributes)

OAM (Object Attribute Memory)

Contains data for 128 OBJs. OAM Size is 512+32 Bytes. The first part (512 bytes) contains 128 4-byte entries for each OBJ:

```
Byte 0 - X-Coordinate (lower 8bit) (upper 1bit at end of OAM)
Byte 1 - Y-Coordinate (all 8bits)
Byte 2 - Tile Number  (lower 8bit) (upper 1bit within Attributes)
Byte 3 - Attributes
```

Attributes:

```
Bit7    Y-Flip (0=Normal, 1=Mirror Vertically)
Bit6    X-Flip (0=Normal, 1=Mirror Horizontally)
Bit5-4  Priority relative to BG (0=Low..3=High)
Bit3-1  Palette Number (0-7) (OBJ Palette 4-7 can use Color Math via CGADSUB)
Bit0    Tile Number (upper 1bit)
```

After above 512 bytes, additional 32 bytes follow, containing 2-bits per OBJ:

```
Bit7    OBJ 3 OBJ Size     (0=Small, 1=Large)
Bit6    OBJ 3 X-Coordinate (upper 1bit)
Bit5    OBJ 2 OBJ Size     (0=Small, 1=Large)
Bit4    OBJ 2 X-Coordinate (upper 1bit)
Bit3    OBJ 1 OBJ Size     (0=Small, 1=Large)
Bit2    OBJ 1 X-Coordinate (upper 1bit)
Bit1    OBJ 0 OBJ Size     (0=Small, 1=Large)
Bit0    OBJ 0 X-Coordinate (upper 1bit)
```

And so on, next 31 bytes with bits for OBJ4..127. Note: The meaning of the OBJ Size bit (Small/Large) can be defined in OBSEL Register (Port 2101h).

SNES PPU Video Memory (VRAM)

BG Map (32x32 entries)

Each BG Map Entry consists of a 16bit value as such:

```
Bit 0-9   - Character Number (000h-3FFh)
Bit 10-12 - Palette Number   (0-7)
Bit 13    - BG Priority      (0=Lower, 1=Higher)
Bit 14    - X-Flip           (0=Normal, 1=Mirror horizontally)
Bit 15    - Y-Flip           (0=Normal, 1=Mirror vertically)
```

In the "Offset-per-Tile" modes (Mode 2,4,6), BG3 entries are different:

```
Bit 15    Apply offset to H/V (0=H, 1=V)  ;-Mode 4 only
Bit 14    Apply offset to BG2             ;\Mode 2 (... and Mode 6, though
Bit 13    Apply offset to BG1             ;/       Mode 6 has only BG1 ?)
Bit 12-10 Not used
Bit 9-0   Scroll offset to be applied to BG1/BG2
Lower 3bit of HORIZONTAL offsets are ignored.
```

In mode7, BG Maps are 128x128, and only the lower 8bit are used as BG map entries:

```
Bit15-8  Not used (contains tile-data; no relation to the BG-Map entry)
Bit7-0   Character Number (00h-FFh) (without XYflip or other attributes)
```

VRAM 8x8 Pixel Tile Data (BG and OBJ)

Each 8x8 tile occupies 16, 32, or 64 bytes (for 4, 16, or 256 colors). BG tiles can be 4/16/256 colors (depending on BG Mode), OBJs are always 16 color.

```
Color Bits (Planes)     Upper Row ........... Lower Row
Plane 0 stored in bytes 00h,02h,04h,06h,08h,0Ah,0Ch,0Eh ;\for 4/16/256 colors
Plane 1 stored in bytes 01h,03h,05h,07h,09h,0Bh,0Dh,0Fh ;/
Plane 2 stored in bytes 10h,12h,14h,16h,18h,1Ah,1Ch,1Eh ;\for 16/256 colors
Plane 3 stored in bytes 11h,13h,15h,17h,19h,1Bh,1Dh,1Fh ;/
Plane 4 stored in bytes 20h,22h,24h,26h,28h,2Ah,2Ch,2Eh ;\
Plane 5 stored in bytes 21h,23h,25h,27h,29h,2Bh,2Dh,2Fh ; for 256 colors
Plane 6 stored in bytes 30h,32h,34h,36h,38h,3Ah,3Ch,3Eh ;
Plane 7 stored in bytes 31h,33h,35h,37h,39h,3Bh,3Dh,3Fh ;/
In each byte, bit7 is left-most, bit0 is right-most.
Plane 0 is the LSB of color number.
```

The only exception are Mode 7 BG Tiles, which are stored as 8bit pixels (without spreading the bits across several bit-planes), and, BG VRAM is divided into BG Map at even byte addresses, and Tiles at odd addresses, an 8x8 tiles thus uses the following bytes (64 odd bytes within a 128 byte region):

```
Vertical Rows           Left-most ..... Right-Most
Upper Row      in bytes 01h,03h,05h,07h,09h,0Bh,0Dh,0Fh ;\
2nd Row        in bytes 11h,13h,15h,17h,19h,1Bh,1Dh,1Fh ;
3rd Row        in bytes 21h,23h,25h,27h,29h,2Bh,2Dh,2Fh ;
4th Row        in bytes 31h,33h,35h,37h,39h,3Bh,3Dh,3Fh ; 256-color
5th Row        in bytes 41h,43h,45h,47h,49h,4Bh,4Dh,4Fh ; Mode 7
```

```
6th Row        in bytes 51h,53h,55h,57h,59h,5Bh,5Dh,5Fh ;
7th Row        in bytes 61h,63h,65h,67h,69h,6Bh,6Dh,6Fh ;
Bottom Row     in bytes 71h,73h,75h,77h,79h,7Bh,7Dh,7Fh ;/
```

16x16 (and bigger) Tiles
BG tiles can be up to 16x16 pixels in size, and OBJs up to 64x64. In both cases, the big tiles are combined of multiple 8x8 pixel tiles, whereas VRAM is organized as "two-dimensional" array of 16x64 BG Tiles and 16x32 OBJ Tiles:
The BG Map or OAM entry contain the Tile number (N) for the upper-left 8x8 tile. The tile(s) right of that tile are N+1 (and N+2, N+3, etc). The tile(s) under that tile are N+10h (and N+20h, N+30h, etc).

```
32x32 pixel OBJ Tile 000h
Tile000h,  Tile001h,  Tile002h,  Tile003h
Tile010h,  Tile011h,  Tile012h,  Tile013h
Tile020h,  Tile021h,  Tile022h,  Tile023h
Tile030h,  Tile031h,  Tile032h,  Tile033h
```

The hex-tile numbers could be thus thought of as "Yyxh", with "x" being the 4bit x-index, and "Yy" being the y-index in the array. For OBJ tiles, the are no carry-outs from "x+1" to "y", nor from "y+1" to "Y". Whilst BG tiles are processing carry-outs. For example:

```
16x16 BG Tile 1FFh     16x16 OBJ Tile 1FFh
Tile1ffh Tile200h      Tile1ffh Tile1f0h
Tile20fh Tile210h      Tile10fh Tile100h
```

Accessing VRAM
SNES Memory VRAM Access (Tile and BG Map)

SNES PPU Color Palette Memory (CGRAM) and Direct Colors

CGRAM Palette Entries

```
15    Not used (should be zero) (read: PPU2 Open Bus)
14-10 Blue
9-5   Green
4-0   Red
```

For accessing CGRAM, see:
SNES Memory CGRAM Access (Palette Memory)

CGRAM Palette Indices

```
00h      Main Backdrop color (used when all BG/OBJ pixels are transparent)
01h-FFh  256-color BG palette (when not using direct-color mode)
01h-7Fh  128-color BG palette (BG2 in Mode 7)
01h-7Fh  Eight 16-color BG palettes
01h-1Fh  Eight 4-color BG palettes (except BG2-4 in Mode 0)
21h-3Fh  Eight 4-color BG palettes (BG2 in Mode 0 only)
41h-5Fh  Eight 4-color BG palettes (BG3 in Mode 0 only)
61h-7Fh  Eight 4-color BG palettes (BG4 in Mode 0 only)
81h-FFh  Eight 16-color OBJ palettes (half of them with color-math disabled)
N/A      Sub Backdrop color (not in CGRAM, set via COLDATA, Port 2132h)
```

Direct Color Mode
256-color BGs (ie. BG1 in Mode 3,4,7) can be optionally set to direct-color mode (via 2130h.Bit0; this bit hides in one of the Color-Math registers, although it isn't related to Color-Math). The 8bit Color number is interpreted as "BBGGGRRR", the 3bit Palette number (from the BG Map) contains LSBs as "bgr", together they are forming a 15bit color "BBb00:RRRr0:GGGg0". Whereas the "bgr" can defined only per tile (not per pixel), and it can be defined only in BG Modes 3-4 (Mode 7 has no palette attributes in BG Map).

```
Color Bit7-0 all zero --> Transparent    ;-Color "Black" is Transparent!
Color Bit7-6   Blue Bit4-3               ;\
Palette Bit 2  Blue Bit2                 ; 5bit Blue
N/A            Blue Bit1-0 (always zero) ;/
Color Bit5-3   Green Bit4-2              ;\
Palette Bit 1  Green Bit1                ; 5bit Green
N/A            Green Bit0 (always zero)  ;/
Color Bit2-0   Red Bit4-2                ;\
Palette Bit 0  Red Bit1                  ; 5bit Red
N/A            Red Bit0 (always zero)    ;/
```

To define Black, either set the Backdrop to Black (works only if there are no layers behind BG1), or use a dark non-transparent "near-black" color (eg. 01h=Dark Red, 09h=Dark Brown).

Screen Border Color
The Screen Border is always Black (no matter of CGRAM settings and Sub Screen Backdrop Color). NTSC 256x224 images are (more or less) fullscreen, so there should be no visible screen border (however, a 2-3 pixel border may appear at the screen edges if the screen isn't properly centered). Both PAL 256x224 and PAL 256x239 images images will have black upper and lower screen borders (a fullscreen PAL picture would be around 256x264, which isn't supported by the SNES).

Forced Blank Color
In Forced Blank, the whole screen is Black (no matter of CGRAM settings, Sub Screen Backdrop Color, and Master Brightness settings). Vsync/Hsync are kept generated (sending a black picture with valid Sync signals to the TV set).

SNES PPU Window

The window feature allows to disable BG/OBJ layers in selected regions, and also to alter Color-Math effects in selected regions.

2126h - WH0 - Window 1 Left Position (X1) (W)
2127h - WH1 - Window 1 Right Position (X2) (W)
2128h - WH2 - Window 2 Left Position (X1) (W)
2129h - WH3 - Window 2 Right Position (X2) (W)
Specifies the horizontal boundaries of the windows. Note that there are no vertical boundaries (these could be implemented by manipulating the window registers via IRQ and/or HDMA).

```
7-0   Window Position (00h..0FFh; 0=leftmost, 255=rightmost)
```

The "inside-window" region extends from X1 to X2 (that, including the X1 and X2 coordinates), so the window width is X2-X1+1. If the width is zero (or negative), then the "inside-window" becomes empty, and the whole screen will be treated "outside-window".

2123h - W12SEL - Window BG1/BG2 Mask Settings (W)

2124h - W34SEL - Window BG3/BG4 Mask Settings (W)
2125h - WOBJSEL - Window OBJ/MATH Mask Settings (W)

```
Bit  2123h 2124h 2125h
7-6  BG2   BG4   MATH  Window-2 Area (0..1=Disable, 1=Inside, 2=Outside)
5-4  BG2   BG4   MATH  Window-1 Area (0..1=Disable, 1=Inside, 2=Outside)
3-2  BG1   BG3   OBJ   Window-2 Area (0..1=Disable, 1=Inside, 2=Outside)
1-0  BG1   BG3   OBJ   Window-1 Area (0..1=Disable, 1=Inside, 2=Outside)
```

Allows to select if the window area is inside or outside the X1,X2 coordinates, or to disable the area.

212Ah/212Bh - WBGLOG/WOBJLOG - Window 1/2 Mask Logic (W)

```
Bit  212Ah 212Bh
7-6  BG4   -     Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
5-4  BG3   -     Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
3-2  BG2   MATH  Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
1-0  BG1   OBJ   Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
```

Allows to merge the Window 1 and 2 areas into a single "final" window area (which is then used by TMW, TSW, and CGWSEL). The OR/AND/XOR/XNOR logic is applied ONLY if BOTH window 1 and 2 are enabled (in WxxSEL registers). If only one window is enabled, then that window is used as is as "final" area. If both are disabled, then the "final" area will be empty. Note: "XNOR" means "1 XOR area1 XOR area2" (ie. the inverse of the normal XOR result).

212Eh - TMW - Window Area Main Screen Disable (W)
212Fh - TSW - Window Area Sub Screen Disable (W)

```
7-5  Not used
4    OBJ (0=Enable, 1=Disable)  ;\"Disable" forcefully disables the layer
3    BG4 (0=Enable, 1=Disable)  ; within the window area (otherwise it is
2    BG3 (0=Enable, 1=Disable)  ; enabled or disabled as selected in the
1    BG2 (0=Enable, 1=Disable)  ; master enable bits in port 212Ch/212Dh)
0    BG1 (0=Enable, 1=Disable)  ;/
-    Backdrop (Always enabled)
```

Allows to disable video layers within the window region.

SNES PPU Color-Math

Main Screen / Sub Screen Enable
Main Screen and Sub Screen BG/OBJ can be enabled via Port 212Ch/212Dh (see PPU Control chapter). When using the Window feature, they can be additionally disabled in selected areas via Port 212Eh/212Fh.
The Backdrops are always enabled as both Main and Sub screen. Of which, the Sub screen backdrop can be effectively disabled (made fully transparent) by setting its color to Black.
The PPU computes the Front-most Non-transparent Main-Screen Pixel, and Front-most Non-transparent Sub-Screen Pixel (or if there's none, it uses the Mainscreen or Subscreen Backdrop color).

2130h - CGWSEL - Color Math Control Register A (W)

```
7-6  Force Main Screen Black (3=Always, 2=MathWindow, 1=NotMathWin, 0=Never)
5-4  Color Math Enable       (0=Always, 1=MathWindow, 2=NotMathWin, 3=Never)
3-2  Not used
1    Sub Screen BG/OBJ Enable    (0=No/Backdrop only, 1=Yes/Backdrop+BG+OBJ)
0    Direct Color (for 256-color BGs)  (0=Use Palette, 1=Direct Color)
```

2131h - CGADSUB - Color Math Control Register B (W)

```
7    Color Math Add/Subtract        (0=Add; Main+Sub, 1=Subtract; Main-Sub)
6    Color Math "Div2" Half Result  (0=No divide, 1=Divide result by 2)
5    Color Math when Main Screen = Backdrop        (0=Off, 1=On) ;\
4    Color Math when Main Screen = OBJ/Palette4..7 (0=Off, 1=On) ; OFF: Show
-    Color Math when Main Screen = OBJ/Palette0..3 (Always=Off)  ; Raw Main,
3    Color Math when Main Screen = BG4             (0=Off, 1=On) ;   or
2    Color Math when Main Screen = BG3             (0=Off, 1=On) ; ON: Show
1    Color Math when Main Screen = BG2             (0=Off, 1=On) ; Main+/-Sub
0    Color Math when Main Screen = BG1             (0=Off, 1=On) ;/
```

Half-Color (Bit6): Ignored if "Force Main Screen Black" is used, also ignored on transparent subscreen pixels (those use the fixed color as sub-screen backdrop without division) (whilst 2130.1 uses the fixed color as non-transparent one, which allows division).
Bit0-5: Seem to affect MAIN SCREEN layers:

```
Disable = Display RAW Main Screen as such (without math)
Enable  = Apply math on Mainscreen
(Ie. 212Ch enables the main screen, 2131h selects if math is applied on it)
```

2132h - COLDATA - Color Math Sub Screen Backdrop Color (W)
This 8bit port allows to manipulate some (or all) bits of a 15bit RGB value. Examples: Black: write E0h (R,G,B=0), Cyan: write 20h (R=0) and DFh (G,B=1Fh).

```
7    Apply Blue  (0=No change, 1=Apply Intensity as Blue)
6    Apply Green (0=No change, 1=Apply Intensity as Green)
5    Apply Red   (0=No change, 1=Apply Intensity as Red)
4-0  Intensity   (0..31)
```

The Sub Screen Backdrop Color is used when all sub screen layers are disabled or transparent, in this case the "Div2" Half Color Math isn't applied (ie. 2131h.Bit6 is ignored); there is one exception: If "Sub Screen BG/OBJ Enable" is off (2130h.Bit1=0), then the "Div2" isn't forcefully ignored.
For a FULLY TRANSPARENT backdrop: Set this register to Black (adding or subtracting black has no effect, and, with "Div2" disabled/ignored, the raw Main screen is displayed as is).

Color Math
Color Math can be disabled by setting 2130h.Bit4-5, or by clearing 2131h.Bit0-5. When it is disabled, only the Main Screen is displayed, and the Sub Screen has no effect on the display.

Color Math occurs only if the front-most Main Screen pixel has math enabled (via 2131h.Bit0-5.), and only if the front-most Sub Screen pixel has same or higher (XXX or is it same or lower -- or is it ANY priority?) priority than the Main Screen pixel.

Same priority means that, for example, BG1 can be mathed with itself: BG1+BG1 gives double-brightness (or same brightness when Div2 is enabled), and, BG1-BG1 gives Black.

Addition/Subtraction is done per R,G,B color fragment, the results can be (optionally) divided by 2, and are then saturated to Max=31 and Min=0).

Force Main Screen Black
This feature forces the whole Main Screen (including Backdrop) to become black, so, normally, the whole screen becomes black. However, color addition can be still applied (but, with the "Div2" not being applied). Whereas, although it looks all black, the Main Screen is still divided into black BG1 pixels, black BG2

pixels, and so on. Of which, one can disable Color Math for some of the pixels. Color Subtraction has no effect (since the pixels can't get blacker than black).

Hires and Pseudo 3-Layer Math
In Hires modes (BG Mode 5,6 and Pseudo Hires via SETINI), the main/sub screen pixels are rendered as half-pixels of the high-resolution image. The TV picture is so blurry, that the result will look quite similar to Color Addition with Div2 - some games (Jurassic Park and Kirby's Dream Land 3) are actually using it for that purpose; the advantage is that one can additionally apply COLDATA addition to (both) main/sub-screen layers, ie. the result looks like "(main+sub)/2+coldata".

SNES PPU Timers and Status

2137h - SLHV - Latch H/V-Counter by Software (R)
```
7-0  Not used (CPU Open Bus; usually last opcode, 21h for "MOV A,[2137h]")
```
Reading from this register latches the current H/V counter values into OPHCT/OPVCT, Ports 213Ch and 213Dh.
Reading here works "as if" dragging IO7 low (but it does NOT actually output a LOW level to IO7 on joypad 2).

213Ch - OPHCT - Horizontal Counter Latch (R)
213Dh - OPVCT - Vertical Counter Latch (R)
There are three situations that do load H/V counter values into the latches:
```
Doing a dummy-read from SLHV (Port 2137h) by software
Switching WRIO (Port 4201h) Bit7 from 1-to-0 by software
Lightgun High-to-Low transition (Pin6 of 2nd Controller connector)
```
All three methods are working only if WRIO.Bit7 is (or was) set. If so, data is latched, and (in all three cases) the latch flag in 213Fh.Bit6 is set.
```
1st read  Lower 8bit
2nd read  Upper 1bit (other 7bit PPU2 open bus; last value read from PPU2)
```
There are two separate 1st/2nd-read flipflops (one for OPHCT, one for OPVCT), both flipflops can be reset by reading from Port 213Fh (STAT78), the flipflops aren't automatically reset when latching occurs.

```
H Counter values range from 0 to 339, with 22-277 being visible on the
screen. V Counter values range from 0 to 261 in NTSC mode (262 is
possible every other frame when interlace is active) and 0 to 311 in
PAL mode (312 in interlace?), with 1-224 (or 1-239(?) if overscan is
enabled) visible on the screen.
```

213Eh - STAT77 - PPU1 Status and Version Number (R)
```
7    OBJ Time overflow  (0=Okay, 1=More than 8x34 OBJ pixels per scanline)
6    OBJ Range overflow (0=Okay, 1=More than 32 OBJs per scanline)
5    Master/Slave Mode (PPU1.Pin25) (0=Normal=Master)
4    Not used (PPU1 open bus) (same as last value read from PPU1)
3-0  PPU1 5C77 Version Number (only version 1 exists as far as I know)
```
The overflow flags are cleared at end of V-Blank, but NOT during forced blank!
The overflow flags are set (regardless of OBJ enable/disable in 212Ch), at following times: Bit6 when V=OBJ.YLOC/H=OAM.INDEX*2, bit7 when V=OBJ.YLOC+1/H=0.

213Fh - STAT78 - PPU2 Status and Version Number (R)
```
7    Current Interlace-Frame (0=1st, 1=2nd Frame)
6    H/V-Counter/Lightgun/Joypad2.Pin6 Latch Flag (0=No, 1=New Data Latched)
5    Not used (PPU2 open bus) (same as last value read from PPU2)
4    Frame Rate (PPU2.Pin30)  (0=NTSC/60Hz, 1=PAL/50Hz)
3-0  PPU2 5C78 Version Number (version 1..3 exist as far as I know)
```
Reading from this register also resets the latch flag (bit6), and resets the two OPHCT/OPVCT 1st/2nd-read flipflops.

Lightgun Coordinates
The screen coordinates (X,Y; with 0,0 = upper left) are related as so:
```
Super Scope  X=OPHCNT-40, Y=OPVCT-1   (games support software calibration)
Justifier 1  X=OPHCNT-??, Y=OPVCT-?   (games support software calibration)
Justifier 2  X=OPHCNT-??, Y=OPVCT-?   (games support software calibration)
M.A.C.S.     X=OPHCNT-76, Y=OPVCT-41  (hardcoded, mechanical calibration)
```
Drawing starts at H=22, V=1 (which explains parts of the offsets). Horizontal offsets greater than 22 are explained by signal switching/transmission delays.
The huge vertical offset of the M.A.C.S. gun is caused by the lightpen being mounted above of the barrel (rather than inside of it) (and apparently not parallel to it, since V>Y instead of V<Y), this implies that the barrel cannot be aimed at screen coordinates Y=151..191 (since the lightpen would be offscreen).
Most games support software calibration. The only exception is M.A.C.S. which requires mechanical hardware calibration (assisted by a test screen, activated by pressing Button A, and then Select Button on joypad 1).
SNES Controllers SuperScope (Lightgun)
SNES Controllers Konami Justifier (Lightgun)
SNES Controllers M.A.C.S. (Lightgun)

SNES PPU Interrupts

4200h - NMITIMEN - Interrupt Enable and Joypad Request (W)
```
7    VBlank NMI Enable  (0=Disable, 1=Enable) (Initially disabled on reset)
6    Not used
5-4  H/V IRQ (0=Disable, 1=At H=H + V=Any, 2=At V=V + H=0, 3=At H=H + V=V)
3-1  Not used
0    Joypad Enable    (0=Disable, 1=Enable Automatic Reading of Joypad)
```
Disabling IRQs (via bit4-5) does additionally acknowledge IRQs. There's no such effect when disabling NMIs (via bit7).

4207h/4208h - HTIMEL/HTIMEH - H-Count Timer Setting (W)
```
15-9  Not used
8-0   H-Count Timer Value (0..339) (+/-1 in long/short lines) (0=leftmost)
```
The H/V-IRQ flag in Bit7 of TIMEUP, Port 4211h gets set when the H-Counter gets equal to the H-Count register value.

4209h/420Ah - VTIMEL/VTIMEH - V-Count Timer Setting (W)
```
15-9  Not used
8-0   V-Count Timer Value (0..261/311, NTSC/PAL) (+1 in interlace) (0=top)
```
The H/V-IRQ flag in Bit7 of TIMEUP, Port 4211h gets set when the V-Counter gets equal to the V-Count register value.

4210h - RDNMI - V-Blank NMI Flag and CPU Version Number (R) (Read/Ack)
```
7    Vblank NMI Flag  (0=None, 1=Interrupt Request) (set on Begin of Vblank)
```

```
6-4   Not used
3-0   CPU 5A22 Version Number (version 2 exists)
```

The NMI flag gets set at begin of Vblank (this happens even if NMIs are disabled). The flag gets reset automatically at end of Vblank, and gets also reset after reading from this register.
The SNES has only one NMI source (vblank), and the NMI flag is automatically reset (on vblank end), so there's normally no need to read/acknowledge the flag, except one special case: If one does disable and re-enable NMIs, then an old NMI may be executed again; acknowledging avoids that effect.
The CPU includes another internal NMI flag, which gets set when "[4200h].7 AND [4210h].7" changes from 0-to-1, and gets cleared when the NMI gets executed (which should happen around after the next opcode) (if a DMA transfer is in progress, then it is somewhere after the DMA, in that case the NMI can get executed outside of the Vblank period, ie. at a time when [4210h].7 is no longer set).

4211h - TIMEUP - H/V-Timer IRQ Flag (R) (Read/Ack)

```
7     H/V-Count Timer IRQ Flag (0=None, 1=Interrupt Request)
6-0   Not used
```

The IRQ flag is automatically reset after reading from this register (except when reading at the very time when the IRQ condition is true (which lasts for 4-8 master cycles), then the CPU receives bit7=1, but register bit7 isn't cleared). The flag is also automatically cleared when disabling IRQs (by setting 4200h.Bit5-4 to zero).
Unlike NMI handlers, IRQ handlers MUST acknowledge IRQs, otherwise the IRQ gets executed again (ie. immediately after the RTI opcode).

4212h - HVBJOY - H/V-Blank flag and Joypad Busy flag (R)

```
7     V-Blank Period Flag (0=No, 1=VBlank)
6     H-Blank Period Flag (0=No, 1=HBlank)
5-1   Not used
0     Auto-Joypad-Read Busy Flag (1=Busy) (see 4200h, and 4218h..421Fh)
```

The Hblank flag gets toggled in ALL scanlines (including during Vblank/Vsync). Both Vblank and Hblank are always toggling (even during Forced Blank, and no matter if IRQs or NMIs are enabled).

Other IRQ Sources
IRQs can be also triggered via Cartridge Slot and Expansion Port. This is done by cartridges with SA-1 and GSU chips.

SNES PPU Resolution

Physical Resolution
The physical resolution (of the TV Screen) is:

```
256x224 for 60Hz (NTSC) consoles
256x264 for 50Hz (PAL) consoles
```

The 50Hz/60Hz cannot be changed by software (it can be only changed by modding some pins on the mainboard).

Normal Picture Resolution
The vertical resolution is software-selectable, 224 or 239 lines:

```
256x224 near-fullscreen on 60Hz (NTSC) consoles (or Tiny Picture at 50Hz)
256x239 not-really-fullscreen on 50Hz (PAL) consoles (or Overscan at 60Hz)
```

Most commonly the resolution is selected matching to the frame rate. With 60Hz Overscan the upper/lower lines are normally not visible (but may be useful to ensure that there is absolutely no upper/lower border visible, which may be important for avoiding flickering in interlace mode, especially with bright pictures; in contrast to the black screen border). 50Hz Tiny Picture would be a simple solution for porting 224-lines NTSC games to PAL, though it produces extra-big upper/lower borders.

High-Resolution Modes
There are some methods to double the resolution horizontally and/or vertically:

True High-Resolution (BG Mode 5,6) (optionally with SETINI.0)
...

Pseudo Horizontal High-Resolution (SETINI.3)
...

Pseudo Vertical High-Resolution (SETINI.0) (Interlace)
...

OBJ High-Resolution (SETINI.1)
...

Hires Notes
Horizontal BG Scrolling is always counted in full-pixels; so true-hires cannot be scrolled in half-pixel units (whilst pseudo hires may be scrolled in half-pixels by exchanging main/sub-screen layers and using different scroll offsets for each layer).
Vertical BG Scrolling is counted in half-pixels (only when using true hv-hires).
Mosaic is always counted in full-pixels; so a mosaic size of 1x1 (which is normally same as mosaic disabled) acts as 2x1 half-pixels in true h-hires mode (and as 2x2 half-pixels in true hv-hires mode, reportedly?) (and as 1x2 in pseudo v-hires, presumably?).
Although h-hires uses main/subscreen for the half-pixels, both main/subscreen pixels are forced to use Color 0 as backdrop color (rather than using COLDATA setting as subscreen backdrop) (this applies for both true+pseudo hires).
Window X1/X2 coordinates are always counted in full-pixels (reportedly with an odd glitch in hires mode?).
OBJ X/Y coordinates are always counted in full-pixels.

Hires Appearance
Horizontal hires may appear blurred on most TV sets (due to the quality of the video signal and TV-screen, and, when not using a RGB-cable, of the composite color clock). For example, non-continous white pixels (W) on black background may appear as gray pixels (g):

```
Source Pixels       --->  TV-Screen Pixels
WW   W   WW   W W         WW   g   WW   ggg
WWW   W   WW   W W        WWW   g   WW   ggg
```

To reproduce that by software: CenterPix=(LeftPix+CenterPix*2+RightPix)/4.
Using the above example on a red background looks even worse: the "g" and "ggg" portions may be barely visible (appear as slightly pastelized red shades).
Vertical hires is producing a flickering image: The scanlines jump up/down after each frame (and, due to a hardware glitch: the topmost lines also jump left/right on PAL consoles). The effect is most annoying with hard-contrast images (eg. black/white text/lines, or bright pixels close to upper/lower screen borders), it may look less annoying with blurry images (eg. photos, or anti-alized text/lines). Some modern TV sets might be buffering the even/odd frames in internal memory, and display them as flicker-free hires image(?)

Hires Software

```
Air Strike Patrol (mission overview)       (whatever mode? with Interlace)
Bishoujo Wrestler Retsuden (some text)     (512x448, BgMode5+Interlace)
Ball Bullet Gun (in lower screen half)     (512x224, BgMode5)
Battle Cross (in game) (but isn't hires?)  (512x224, BgMode1+PseudoH)(Bug?)
BS Radical Dreamers (user name input only) (512x224, BgMode5)
Chrono Trigger (crash into Lavos sequence) (whatever mode? with Interlace)
Donkey Kong Country 1 (Nintendo logo)      (512x224, BgMode5)
G.O.D. (intro & lower screen half)         (512x224, BgMode5)
Jurassic Park (score text)                 (512x224, BgMode1+PseudoH+Math)
Kirby's Dream Land 3 (leaves in 1st door)  (512x224, BgMode1+PseudoH)
Lufia 2 (credits screen at end of game)    (whatever mode?)
Moryo Senki Madara 2 (text)                (512x224, BgMode5)
Power Drive (in intro)                     (512x448, BgMode5+Interlace)
Ranma 1/2: Chounai Gekitou Hen             (256x448, BgMode1+InterlaceBug)
RPM Racing (in intro and in game)          (512x448, BgMode5+Interlace)
Rudra no Hihou (RnH/Treasure of the Rudras)(512x224, BgMode5)
Seiken Densetsu 2 (Secret of Mana) (setup) (512x224, BgMode5)
Seiken Densetsu 3                          (512x224, BgMode5)
Shock Issue 1 & 2 (homebrew eZine)         (512x224, BgMode5)
SNES Test Program (by Nintendo) (Character Test includes BgMode5/BgMode6)
Super Play Action Football (text)          (512x224, BgMode5)
World Cup Striker (intro/menu)             (512x224, BgMode5)
```

And, reportedly (may be incorrect, unknown how to see hires in that games):

```
Dragonball Z Super Butoden 2-3 (when you start it up in the black screen?)
```

Notes: Ranma is actually only 256x224 (but does accidentally have interlace enabled, which causes some totally useless flickering). Jurassic/Kirby are misusing PseudoH for "blending" the main+sub screen layers (via the blurry TV picture); this allows to use Color Math for coldata additions (ie. resulting in 3-color blending: main+sub+coldata).

SNES PPU Offset-Per-Tile Mode

XXX - Under construction (see Anomie's docs for now)

Offset-Per-Tile Mode is used by following Programs

```
Chrono Trigger (title screen, intro's "Black Omen" appearing)
Star Fox/Starwing (to "rotate" the landscape background)
Tetris Attack
Yoshi's Island (dizziness effect, wavy lava)
```

SNES Audio Processing Unit (APU)

Overview
SNES APU Memory and I/O Map
SNES APU Block Diagram

SPC700 CPU
SNES APU SPC700 CPU Overview
SNES APU SPC700 CPU Load/Store Commands
SNES APU SPC700 CPU ALU Commands
SNES APU SPC700 CPU Jump/Control Commands

I/O Ports
SNES APU SPC700 I/O Ports
SNES APU Main CPU Communication Port
SNES APU DSP BRR Samples
SNES APU DSP BRR Pitch
SNES APU DSP ADSR/Gain Envelope
SNES APU DSP Volume Registers
SNES APU DSP Control Registers
SNES APU DSP Echo Registers

Pinouts/Misc
SNES APU Low Level Timings
SNES Audio/Video Connector Pinouts
SNES Pinouts APU Chips

SNES APU Memory and I/O Map

SPC700 Memory Map

```
0000h..00EFh  RAM (typically used for CPU pointers/variables)
00F0h..00FFh  I/O Ports (writes are also passed to RAM)
0100h..01FFh  RAM (typically used for CPU stack)
0200h..FFBFh  RAM (code, data, dir-table, brr-samples, echo-buffer, etc.)
FFC0h..FFFFh  64-byte Boot ROM or RAM (selectable via Port 00F1h)
```

Audio-related Registers on Main CPU
The main CPU has four write-only 8bit outputs, and (mapped to the same addresses) four read-only 8bit inputs:

```
2140h - APUI00  - Main CPU to Sound CPU Communication Port 0
2141h - APUI01  - Main CPU to Sound CPU Communication Port 1
2142h - APUI02  - Main CPU to Sound CPU Communication Port 2
2143h - APUI03  - Main CPU to Sound CPU Communication Port 3
```

The registers are used to communicate with Port 00F4h..00F7h on the SPC700 CPU (on power-up, this is done using a 64-byte BIOS in the SPC700).
Note: All CPU-APU communications are passed through these registers by software, there are no additional communication methods (like IRQs).

SPC700 I/O Ports
The SPC700 CPU includes 16 memory mapper ports at address 00F0h..00FFh:

```
00F0h - TEST    - Testing functions (W)                          0Ah
00F1h - CONTROL - Timer, I/O and ROM Control (W)                 80h
```

```
00F2h - DSPADDR - DSP Register Index (R/W)                          (FFh)
00F3h - DSPDATA - DSP Register Data (R/W)                         (DSP[7Fh])
00F4h - CPUIO0  - CPU Input and Output Register 0 (R and W)      R=00h,W=00h
00F5h - CPUIO1  - CPU Input and Output Register 1 (R and W)      R=00h,W=00h
00F6h - CPUIO2  - CPU Input and Output Register 2 (R and W)      R=00h,W=00h
00F7h - CPUIO3  - CPU Input and Output Register 3 (R and W)      R=00h,W=00h
00F8h - AUXIO4  - External I/O Port P4 (S-SMP Pins 34-27) (R/W) (unused) FFh
00F9h - AUXIO5  - External I/O Port P5 (S-SMP Pins 25-18) (R/W) (unused) FFh
00FAh - T0DIV   - Timer 0 Divider (for 8000Hz clock source) (W)         (FFh)
00FBh - T1DIV   - Timer 1 Divider (for 8000Hz clock source) (W)         (FFh)
00FCh - T2DIV   - Timer 2 Divider (for 64000Hz clock source) (W)        (FFh)
00FDh - T0OUT   - Timer 0 Output (R)                                    (00h)
00FEh - T1OUT   - Timer 1 Output (R)                                    (00h)
00FFh - T2OUT   - Timer 2 Output (R)                                    (00h)
```

DSP Registers

The 128 DSP Registers are indirectly accessed via SPC700 Ports 00F2h/00F3h.
("x" can be 0..7, for selecting one of the 8 voices, or of the 8 filters).

```
x0h - VxVOLL   - Left volume for Voice 0..7 (R/W)
x1h - VxVOLR   - Right volume for Voice 0..7 (R/W)
x2h - VxPITCHL - Pitch scaler for Voice 0..7, lower 8bit (R/W)
x3h - VxPITCHH - Pitch scaler for Voice 0..7, upper 6bit (R/W)
x4h - VxSRCN   - Source number for Voice 0..7 (R/W)
x5h - VxADSR1  - ADSR settings for Voice 0..7, lower 8bit (R/W)
x6h - VxADSR2  - ADSR settings for Voice 0..7, upper 8bit (R/W)
x7h - VxGAIN   - Gain settings for Voice 0..7 (R/W)
x8h - VxENVX   - Current envelope value for Voice 0..7 (R)
x9h - VxOUTX   - Current sample value for Voice 0..7 (R)
xAh - NA       - Unused (8 bytes of general-purpose RAM) (R/W)
xBh - NA       - Unused (8 bytes of general-purpose RAM) (R/W)
0Ch - MVOLL    - Left channel master volume (R/W)
1Ch - MVOLR    - Right channel master volume (R/W)
2Ch - EVOLL    - Left channel echo volume (R/W)
3Ch - EVOLR    - Right channel echo volume (R/W)
4Ch - KON      - Key On Flags for Voice 0..7 (W)
5Ch - KOFF     - Key Off Flags for Voice 0..7 (R/W)
6Ch - FLG      - Reset, Mute, Echo-Write flags and Noise Clock (R/W)
7Ch - ENDX     - Voice End Flags for Voice 0..7 (R) (W=Ack)
0Dh - EFB      - Echo feedback volume (R/W)
1Dh - NA       - Unused (1 byte of general-purpose RAM) (R/W)
2Dh - PMON     - Pitch Modulation Enable Flags for Voice 1..7 (R/W)
3Dh - NON      - Noise Enable Flags for Voice 0..7 (R/W)
4Dh - EON      - Echo Enable Flags for Voice 0..7 (R/W)
5Dh - DIR      - Sample table address (R/W)
6Dh - ESA      - Echo ring buffer address (R/W)
7Dh - EDL      - Echo delay (ring buffer size) (R/W)
xEh - NA       - Unused (8 bytes of general-purpose RAM) (R/W)
xFh - FIRx     - Echo FIR filter coefficient 0..7 (R/W)
```

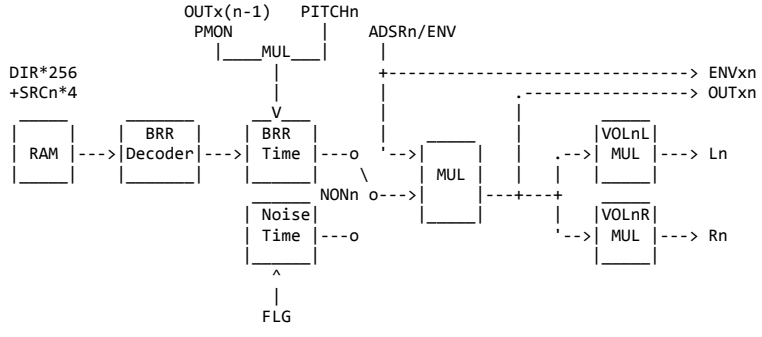
Register 80h..FFh are read-only mirrors of 00h..7Fh.
Technically, the DSP registers are a RAM-like 128-byte buffer; and are copied to internal registers when needed. At least some registers (KON and FLG) seem to have changed-flags (and the buffer-values are processed only when the flag is set), ENDX seems to be a real register (not using the RAM-like buffer).
Upon Reset, FLG is internally set to E0h (but reading the buffered DSP[6Ch] value return garbage. The 17 status register are updated (a few cycles after reset) to ENDX=FFh, ENVx=00h, OUTx=00h. Aside from ENDX/ENVx/OUTx, all other DSP registers contain garbage. Upon Power-up, that garbage randomly "tends" to some patterns, but those patterns change from day to day (some examples: one day all registers were set to 43h, on other days, the eight FIRx were all FFh, and other registers had whichever values).

APU RAM

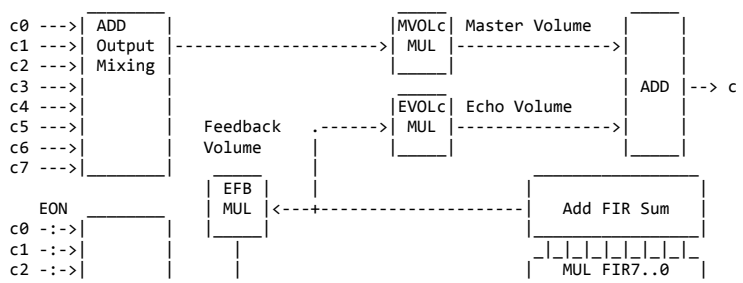
Upon power-up, APU RAM tends to contain a stable repeating 64-byte pattern: 32x00h, 32xFFh (that, for APUs with two Motorola MCM51L832F12 32Kx8 SRAM chips; other consoles may use different chips with different garbage/patterns). After Reset, the boot ROM changes [0000h..0001h]=Entrypoint, and [0002h..00EFh]=00h).

SNES APU Block Diagram

DSP Voice Block Diagram (n=voice, 0..7)



DSP Mixer/Reverb Block Diagram (c=channel, L/R)



```
c3 -:->|        |      |         ESA=Addr, EDL=Len   |_7_6_5_4_3_2_1_0_|
c4 -:->| ADD    |     __V__  FLG  _______________      _|_|_|_|_|_|_|_|_
c5 -:->| Echo   |    |     | ECEN | Echo Buffer c |    | FIR Buffer c    |
c6 -:->| Mixing |-->| ADD |--:-->|   RAM         |-->| (Hardware regs) |
c7 -:->|________|   |_____|      |_______________|    |_________________|
                                 Newest --> Oldest     Newest --> Oldest
```

External Sound Output & Input

```
 _____    _____    ____________    _______    _______
|     |  |     |  | Pre-       |  |       |  | Post- |
| DSP |->| D/A |->| Amplifier  |->| Analog|->| Ampl. |--> Multi-Out
|_____|  |_____|  |____________|  | Mixer |  |       |    (Stereo Out)
   |                               |       |  | (with |
   |   Cartridge Slot Stereo In --->|       |  | phase |--> TV Modulator
   |                               |       |  | inver-|    (Mono Out)
   |   Expansion Port Stereo In --->|       |  | sion) |
   |                               |_______|  |       |--> Expansion Port
   |   /MUTE signal                    |      |       |    (Mono Out)
   '---------------------------------->----'    |_______|
```

Note: The Cartridge Audio input is used only by the Super Gameboy. Unknown if it is also used by X-Band modem (for injecting dial/connection sounds)? The Expansion port input might be used by the Satellaview (?)
The Nintendo Super System (NSS) also allows to mute the SNES sound via its Z80 CPU.

SNES APU SPC700 CPU Overview

The sound unit is controlled by a S-SMP chip: an 8bit CPU that uses Sony's SPC700 instruction set. The SPC700's opcodes and it's three main registers (A,X,Y) are clearly inspired by 6502 instruction set, although the opcodes are numbered differently, and it has some new/changed instructions.
Aside from the SNES sound processor, the SPC700 is also found in Sony's CXP8nnnn single-chip microprocessors. Another variant, called SPC700 alpha II, is also found in their CXP7nnnn chips, which have only 211 opcodes (two less than the normal SPC700).

Registers

```
A     8bit accumulator
X     8bit index
Y     8bit index
SP    8bit stack pointer (addresses memory at 0100h..01FFh)
PSW   8bit flags
YA    16bit combination of Y=MSB, and A=LSB
PC    16bit program counter
```

Flags (PSW)

```
Bit7  N  Sign Flag          (0=Positive, 1=Negative)
Bit6  V  Overflow Flag      (0=None, 1=Overflow)
Bit5  P  Zero Page Location (0=00xxh, 1=01xxh)
Bit4  B  Break Flag         (0=Reset, 1=BRK opcode; set <after> BRK opcode)
Bit3  H  Half-carry         (0=Borrow, or no-carry, 1=Carry, or no-borrow)
Bit2  I  Interrupt Enable   (0=Disable, 1=Enable) (no function in SNES APU)
Bit1  Z  Zero Flag          (0=Non-zero, 1=Zero)
Bit0  C  Carry Flag         (0=Borrow, or no-carry, 1=Carry, or no-borrow)
```

Addressing Modes

```
Native Syntax  Nocash Syntax
aa             [aa]           ;\addresses memory at [0000..00FF]
aa+X           [aa+X]         ; (or at [0100..01FF when flag P=1)
aa+Y           [aa+Y]         ; (aa+X and aa+Y wrap within that area,
(X)            [X]            ; ie. addr = (aa+X) AND 0FFh)
(Y)            [Y]            ;/
aaaa           [aaaa]         ;\
aaaa+X         [aaaa+X]       ; addresses memory at [0000..FFFF]
aaaa+Y         [aaaa+Y]       ;/
[aa]+Y         [[aa]+Y]       ;-Byte[Word[aa]+Y]  ;\double-indirect (using
[aa+X]         [[aa+X]]       ;-Byte[Word[aa+X]]  ;/16bit pointer in RAM)
aa.b           [aa].b         ;-Bit0..7 of address [0000..00FF] (8bit addr)
aaa.b          [aaa].b        ;-Bit0..7 of address [0000..1FFF] (13bit addr)
stack (push/pop/call/ret)     ;-addresses memory at [0100..01FF] (SP+100h)
```

Formatting of Command Descriptions

```
Native Syntax  Nocash Syntax   Opcode     Clk Expl                  NVPBHIZC
```

SNES APU SPC700 CPU Load/Store Commands

Register Manipulation

```
MOV  A,#nn     MOV  A,nn       E8 nn       2  A=nn                 N.....Z.
MOV  X,#nn     MOV  X,nn       CD nn       2  X=nn                 N.....Z.
MOV  Y,#nn     MOV  Y,nn       8D nn       2  Y=nn                 N.....Z.
MOV  A,X       MOV  A,X        7D          2  A=X                  N.....Z.
MOV  X,A       MOV  X,A        5D          2  X=A                  N.....Z.
MOV  A,Y       MOV  A,Y        DD          2  A=Y                  N.....Z.
MOV  Y,A       MOV  Y,A        FD          2  Y=A                  N.....Z.
MOV  X,SP      MOV  X,SP       9D          2  X=SP                 N.....Z.
MOV  SP,X      MOV  SP,X       BD          2  SP=X  ;at 0100..01FF  ........
```

Memory Load

```
MOV  A,aa      MOV  A,[aa]     E4 aa       3  A=[aa]               N.....Z.
MOV  A,aa+X    MOV  A,[aa+X]   F4 aa       4  A=[aa+X]             N.....Z.
MOV  A,!aaaa   MOV  A,[aaaa]   E5 aa aa    4  A=[aaaa]             N.....Z.
MOV  A,!aaaa+X MOV  A,[aaaa+X] F5 aa aa    5  A=[aaaa+X]           N.....Z.
MOV  A,!aaaa+Y MOV  A,[aaaa+Y] F6 aa aa    5  A=[aaaa+Y]           N.....Z.
MOV  A,(X)     MOV  A,[X]      E6          3  A=[X]                N.....Z.
MOV  A,(X)+    MOV  A,[X]+     BF          4  A=[X], X=X+1         N.....Z.
MOV  A,[aa]+Y  MOV  A,[[aa]+Y] F7 aa       6  A=[[aa]+Y]           N.....Z.
MOV  A,[aa+X]  MOV  A,[[aa+X]] E7 aa       6  A=[[aa+X]]           N.....Z.
MOV  X,aa      MOV  X,[aa]     F8 aa       3  X=[aa]               N.....Z.
```

```
MOV  X,aa+Y    MOV  X,[aa+Y]   F9 aa       4  X=[aa+Y]              N.....Z.
MOV  X,!aaaa   MOV  X,[aaaa]   E9 aa aa    4  X=[aaaa]              N.....Z.
MOV  Y,aa      MOV  Y,[aa]     EB aa       3  Y=[aa]                N.....Z.
MOV  Y,aa+X    MOV  Y,[aa+X]   FB aa       4  Y=[aa+X]              N.....Z.
MOV  Y,!aaaa   MOV  Y,[aaaa]   EC aa aa    4  Y=[aaaa]              N.....Z.
MOVW YA,aa     MOVW YA,[aa]    BA aa       5  YA=Word[aa]            N.....Z.
```

Memory Store

```
MOV  aa,#nn    MOV  [aa],nn    8F nn aa    5  [aa]=nn      (read)    ........
MOV  aa,bb     MOV  [aa],[bb]  FA bb aa    5  [aa]=[bb]    (no read) ........
MOV  aa,A      MOV  [aa],A     C4 aa       4  [aa]=A       (read)    ........
MOV  aa,X      MOV  [aa],X     D8 aa       4  [aa]=X       (read)    ........
MOV  aa,Y      MOV  [aa],Y     CB aa       4  [aa]=Y       (read)    ........
MOV  aa+X,A    MOV  [aa+X],A   D4 aa       5  [aa+X]=A     (read)    ........
MOV  aa+X,Y    MOV  [aa+X],Y   DB aa       5  [aa+X]=Y     (read)    ........
MOV  aa+Y,X    MOV  [aa+Y],X   D9 aa       5  [aa+Y]=X     (read)    ........
MOV  !aaaa,A   MOV  [aaaa],A   C5 aa aa    5  [aaaa]=A     (read)    ........
MOV  !aaaa,X   MOV  [aaaa],X   C9 aa aa    5  [aaaa]=X     (read)    ........
MOV  !aaaa,Y   MOV  [aaaa],Y   CC aa aa    5  [aaaa]=Y     (read)    ........
MOV  !aaaa+X,A MOV  [aaaa+X],A D5 aa aa    6  [aaaa+X]=A   (read)    ........
MOV  !aaaa+Y,A MOV  [aaaa+Y],A D6 aa aa    6  [aaaa+Y]=A   (read)    ........
MOV  (X)+,A    MOV  [X]+,A     AF          4  [X]=A, X=X+1 (no read) ........
MOV  (X),A     MOV  [X],A      C6          4  [X]=A        (read)    ........
MOV  [aa]+Y,A  MOV  [[aa]+Y],A D7 aa       7  [[aa]+Y]=A   (read)    ........
MOV  [aa+X],A  MOV  [[aa+X]],A C7 aa       7  [[aa+X]]=A   (read)    ........
MOVW aa,YA     MOVW [aa],YA    DA aa       5  Word[aa]=YA  (read lsb)........
```

Most of the Memory Store opcodes are implemented like ALU opcodes (ie. as RMW opcodes, issuing a dummy read from the destination address). In result, they are a bit slower than necessary, and they can trigger read-sensitive I/O ports (namely the TnOUT registers in the SNES). Only exceptions are opcodes AFh and FAh (which don't include any dummy read), and opcode DAh (which performs the dummy read only on the LSB of the 16bit Word).

Push/Pop

```
PUSH A         PUSH A          2D          4  [SP]=A,     SP=SP-1    ........
PUSH X         PUSH X          4D          4  [SP]=X,     SP=SP-1    ........
PUSH Y         PUSH Y          6D          4  [SP]=Y,     SP=SP-1    ........
PUSH PSW       PUSH PSW        0D          4  [SP]=Flags, SP=SP-1    ........
POP  A         POP  A          AE          4  SP=SP+1, A=[SP]        ........
POP  X         POP  X          CE          4  SP=SP+1, X=[SP]        ........
POP  Y         POP  Y          EE          4  SP=SP+1, Y=[SP]        ........
POP  PSW       POP  PSW        8E          4  SP=SP+1, Flags=[SP]    NVPBHIZC
```

SNES APU SPC700 CPU ALU Commands

8bit ALU Operations

```
OR   a,b       OR   a,b        00+x ...    .. a=a OR b              N.....Z.
AND  a,b       AND  a,b        20+x ...    .. a=a AND b             N.....Z.
EOR  a,b       XOR  a,b        40+x ...    .. a=a XOR b             N.....Z.
CMP  a,b       CMP  a,b        60+x ...    .. a-b                   N.....ZC
ADC  a,b       ADC  a,b        80+x ...    .. a=a+b+C               NV..H.ZC
SBC  a,b       SBC  a,b        A0+x ...    .. a=a-b-not C           NV..H.ZC
```

Above OR/AND/EOR/CMP/ADC/SBC can be used with following operands:

```
cmd  A,#nn     cmd  A,nn       x+08 nn     2  A,nn
cmd  A,(X)     cmd  A,[X]      x+06        3  A,[X]
cmd  A,aa      cmd  A,[aa]     x+04 aa     3  A,[aa]
cmd  A,aa+X    cmd  A,[aa+X]   x+14 aa     4  A,[aa+X]
cmd  A,!aaaa   cmd  A,[aaaa]   x+05 aa aa  4  A,[aaaa]
cmd  A,!aaaa+X cmd  A,[aaaa+X] x+15 aa aa  5  A,[aaaa+X]
cmd  A,!aaaa+Y cmd  A,[aaaa+Y] x+16 aa aa  5  A,[aaaa+Y]
cmd  A,[aa]+Y  cmd  A,[[aa]+Y] x+17 aa     6  A,[[aa]+Y]
cmd  A,[aa+X]  cmd  A,[[aa+X]] x+07 aa     6  A,[[aa+X]]
cmd  aa,bb     cmd  [aa],[bb]  x+09 bb aa  6  [aa],[bb]
cmd  aa,#nn    cmd  [aa],nn    x+18 nn aa  5  [aa],nn
cmd  (X),(Y)   cmd  [X],[Y]    x+19        5  [X],[Y]
```

Compare can additionally have the following forms:

```
CMP  X,#nn     CMP  X,nn       C8 nn       2  X-nn                  N.....ZC
CMP  X,aa      CMP  X,[aa]     3E aa       3  X-[aa]                 N.....ZC
CMP  X,!aaaa   CMP  X,[aaaa]   1E aa aa    4  X-[aaaa]               N.....ZC
CMP  Y,#nn     CMP  Y,nn       AD nn       2  Y-nn                  N.....ZC
CMP  Y,aa      CMP  Y,[aa]     7E aa       3  Y-[aa]                 N.....ZC
CMP  Y,!aaaa   CMP  Y,[aaaa]   5E aa aa    4  Y-[aaaa]               N.....ZC
```

Note: There's also a compare and jump if non-zero command (see jumps).

8bit Increment/Decrement and Rotate/Shift Commands

```
ASL  a         SHL  a          00+x ..     .. Left shift, bit0=0    N.....ZC
ROL  a         RCL  a          20+x ..     .. Left shift, bit0=C    N.....ZC
LSR  a         SHR  a          40+x ..     .. Right shift, bit7=0   N.....ZC
ROR  a         RCR  a          60+x ..     .. Right shift, bit7=C   N.....ZC
DEC  a         DEC  a          80+x ..     .. a=a-1                 N.....Z.
INC  a         INC  a          A0+x ..     .. a=a+1                 N.....Z.
```

The Increment/Decrement and Rotate/Shift commands can have following forms:

```
cmd  A         cmd  A          x+1C        2  A
cmd  X         cmd  X          x+1D-80     2  X    ;\increment/decrement only
cmd  Y         cmd  Y          x+DC-80     2  Y    ;/(not rotate/shift)
cmd  aa        cmd  [aa]       x+0B aa     4  [aa]
cmd  aa+X      cmd  [aa+X]     x+1B aa     5  [aa+X]
cmd  !aaaa     cmd  [aaaa]     x+0C aa aa  5  [aaaa]
```

Note: There's also a decrement and jump if non-zero command (see jumps).

16bit ALU Operations

```
ADDW YA,aa     ADDW YA,[aa]    7A aa       5  YA=YA+Word[aa]         NV..H.ZC
SUBW YA,aa     SUBW YA,[aa]    9A aa       5  YA=YA-Word[aa]         NV..H.ZC
CMPW YA,aa     CMPW YA,[aa]    5A aa       4  YA-Word[aa]            N.....ZC
INCW aa        INCW [aa]       3A aa       6  Word[aa]=Word[aa]+1    N.....Z.
DECW aa        DECW [aa]       1A aa       6  Word[aa]=Word[aa]-1    N.....Z.
DIV  YA,X      DIV  YA,X       9E         12  A=YA/X, Y=YA MOD X     NV..H.Z.
MUL  YA        MUL  YA         CF          9  YA=Y*A, NZ on Y only   N.....Z.
```

For ADDW/SUBW, H is carry from bit11 to bit12.

1bit ALU Operations

```
CLR1 aa.b      CLR  [aa].b     b*20+12 aa  4  [aa].bit_b=0           ........
SET1 aa.b      SET  [aa].b     b*20+02 aa  4  [aa].bit_b=1           ........
NOT1 aaa.b     NOT  [aaa].b    EA aa ba    5  invert [aaa].bit_b     ........
MOV1 aaa.b,C   MOV  [aaa].b,C  CA aa ba    6  [aaa].bit_b=C          ........
MOV1 C,aaa.b   MOV  C,[aaa].b  AA aa ba    4  C=[aaa].bit_b          .......C
OR1  C,aaa.b   OR   C,[aaa].b  0A aa ba    5  C=C OR [aaa].bit_b     .......C
OR1  C,/aaa.b  OR   C,not[].b  2A aa ba    5  C=C OR not[aaa].bit_b  .......C
AND1 C,aaa.b   AND  C,[aaa].b  4A aa ba    4  C=C AND [aaa].bit_b    .......C
AND1 C,/aaa.b  AND  C,not[].b  6A aa ba    4  C=C AND not[aaa].bit_b .......C
EOR1 C,aaa.b   XOR  C,[aaa].b  8A aa ba    5  C=C XOR [aaa].bit_b    .......C
CLRC           CLR  C          60          2  C=0                    .......0
SETC           SET  C          80          2  C=1                    .......1
NOTC           NOT  C          ED          3  C=not C                .......C
CLRV           CLR  V,H        E0          2  V=0, H=0               .0..0...
```

Special ALU Operations

```
DAA   A        DAA  A          DF          3  BCD adjust after ADC   N.....ZC
DAS   A        DAS  A          BE          3  BCD adjust after SBC   N.....ZC
XCN   A        XCN  A          9F          5  A = (A>>4) | (A<<4)    N.....Z.
TCLR1 !aaaa    TCLR [aaaa],A   4E aa aa     6  [aaaa]=[aaaa]AND NOT A N.....Z.
TSET1 !aaaa    TSET [aaaa],A   0E aa aa     6  [aaaa]=[aaaa]OR A      N.....Z.
```

For TCLR/TSET, Z+N flags are set as for "CMP A,[aaaa]" (before [aaaa] gets changed).
Reportedly, TCLR/TSET can access only 0000h..7FFFh, that is nonsense.

SNES APU SPC700 CPU Jump/Control Commands

Conditional Jumps

```
BPL dest       JNS dest        10 rr         2/4  if N=0 --> JR dest ........
BMI dest       JS  dest        30 rr         2/4  if N=1 --> JR dest ........
BVC dest       JNO dest        50 rr         2/4  if V=0 --> JR dest ........
BVS dest       JO  dest        70 rr         2/4  if V=1 --> JR dest ........
BCC dest       JNC dest        90 rr         2/4  if C=0 --> JR dest ........
BCS dest       JC  dest        B0 rr         2/4  if C=1 --> JR dest ........
BNE dest       JNZ dest        D0 rr         2/4  if Z=0 --> JR dest ........
BEQ dest       JZ  dest        F0 rr         2/4  if Z=1 --> JR dest ........
BBS aa.b,dest  JNZ [aa].b,dest b*20+03 aa rr 5/7  if [aa].bit_b=1 -> ........
BBC aa.b,dest  JZ  [aa].b,dest b*20+13 aa rr 5/7  if [aa].bit_b=0 -> ........
CBNE aa,dest   CJNE A,[aa],d   2E aa rr      5/7  if A<>[aa]   -->   ........
CBNE aa+X,dest CJNE A,[aa+X],d DE aa rr      6/8  if A<>[aa+X] -->   ........
DBNZ Y,dest    DJNZ Y,dest     FE rr         4/6  Y=Y-1, if Y<>0 --> ........
DBNZ aa,dest   DJNZ [aa],dest  6E aa rr      5/7  [aa]=[aa]-1, if .. ........
```

Aliases: JZ=JE, JNZ=JNE, JC=JAE, JNC=JB.

Normal Jumps/Calls

```
BRA  dest      JR   dest       2F rr       4  PC=PC+/-rr             ........
JMP  !aaaa     JMP  aaaa       5F aa aa    3  PC=aaaa                ........
JMP  [!aaaa+X] JMP  [aaaa+X]   1F aa aa    6  PC=Word[a+X]           ........
CALL !aaaa     CALL aaaa       3F aa aa    8  [S-1]=PC,S=S-2,PC=aaaa ........
TCALL n        CALL [FFnn]     n1 ;n=0..F  8  Push PC, PC=[FFDE-n*2] ........
PCALL uu       PCALL FFnn      4F nn       6  Push PC, PC=FF00..FFFF ........
RET            RET             6F          5  PC=[S+1],S=S+2         ........
RET1           RETI            7F          6  Pop Flags, PC         NVPBHIZC
BRK            BRK             0F          8  Push $+1,PSW,PC=[FFDE] ...1.0..
/RESET         /RESET          -           ?  PC=[FFFEh]             ..00.0..
```

Note: Calls are pushing the exact retadr (unlike 6502, which pushes retadr-1).

Wait/Delay/Control

```
NOP            NOP             00          2  do nothing             ........
SLEEP          SLEEP           EF          ?  Halts the processor    ........
STOP           STOP            FF          ?  Halts the processor    ........
CLRP           CLR P           20          2  P=0 ;zero page at 00aa ..0.....
SETP           SET P           40          2  P=1 ;zero page at 01aa ..1.....
EI             EI              A0          3  I=1 ;interrupt enable  .....1..
DI             DI              C0          3  I=0 ;interrupt disable .....0..
```

Note: The SNES APU doesn't have any interrupt sources, so SLEEP/STOP will hang the CPU forever, and DI/EI have no effect (other than changing to I-flag in PSW).

SNES APU SPC700 I/O Ports

00F0h - TEST - Testing functions (W)

```
0    Timer-Enable     (0=Normal, 1=Timers don't work)
1    RAM Write Enable (0=Disable/Read-only, 1=Enable SPC700 & S-DSP writes)
2    Crash SPC700     (0=Normal, 1=Crashes the CPU)
3    Timer-Disable    (0=Timers don't work, 1=Normal)
4-5  Waitstates on RAM Access         (0..3 = 0/1/4/9 cycles) (0=Normal)
6-7  Waitstates on I/O and ROM Access (0..3 = 0/1/4/9 cycles) (0=Normal)
```

Default setting is 0Ah, software should never change this register. Normal memory access time is 1 cycle (adding 0/1/4/9 waits gives access times of 1/2/5/10 cycles). Using 4 or 9 waits doesn't work with some opcodes (0 or 1 waits seem to work stable).
Internal cycles (those that do not access RAM, ROM, nor I/O) are either using the RAM or I/O access time (see notes at bottom of this chapter).

00F1h - CONTROL - Timer, I/O and ROM Control (W)

```
0-2  Timer 0-2 Enable (0=Disable, set TnOUT=0 & reload divider, 1=Enable)
3    Not used
4    Reset Port 00F4h/00F5h Input-Latches (0=No change, 1=Reset to 00h)
5    Reset Port 00F6h/00F7h Input-Latches (0=No change, 1=Reset to 00h)
      Note: The CPUIO inputs are latched inside of the SPC700 (at time when
      the Main CPU writes them), above two bits allow to reset these latches.
6    Not used
7    ROM at FFC0h-FFFFh (0=RAM, 1=ROM) (writes do always go to RAM)
```

On power on or reset, it seems to be set to #$B0.

00F2h - DSPADDR - DSP Register Index (R/W)
```
0-7  DSP Register Number (usually 00h..7Fh) (80h..FFh are read-only mirrors)
```

00F3h - DSPDATA - DSP Register Data (R/W)
```
0-7  DSP Register Data (read/write the register selected via Port 00F2h)
```

00F4h - CPUIO0 - CPU Input and Output Register 0 (R and W)
00F5h - CPUIO1 - CPU Input and Output Register 1 (R and W)
00F6h - CPUIO2 - CPU Input and Output Register 2 (R and W)
00F7h - CPUIO3 - CPU Input and Output Register 3 (R and W)
These registers are used in communication with the 5A22 S-CPU. There are eight total registers accessed by these four addresses: four write-only output ports to the S-CPU and four read-only input ports from the S-CPU.
```
0-7  Data written to/read from corresponding register on main 5A22 CPU
```
If the SPC700 writes to an output port while the S-CPU is reading it, the S-CPU will read the logical OR of the old and new values. Possibly the same thing happens the other way around, but the details are unknown?

00F8h - AUXIO4 - External I/O Port P4 (S-SMP Pins 34-27) (R/W) (unused)
00F9h - AUXIO5 - External I/O Port P5 (S-SMP Pins 25-18) (R/W) (unused)
Writing changes the output levels. Reading normally returns the same value as the written value; unless external hardware has pulled the pins LOW or HIGH. Reading from AUXIO5 additionally produces a short /P5RD read signal (Pin17).
```
0-7  Input/Output levels (0=Low, 1=High)
```
In the SNES, these pins are unused (not connected), so the registers do effectively work as if they'd be "RAM-like" general purpose storage registers.

00FAh - T0DIV - Timer 0 Divider (for 8000Hz clock source) (W)
00FBh - T1DIV - Timer 1 Divider (for 8000Hz clock source) (W)
00FCh - T2DIV - Timer 2 Divider (for 64000Hz clock source) (W)
```
0-7  Divider (01h..FFh=Divide by 1..255, or 00h=Divide by 256)
```
If timers are enabled (via Port 00F1h), then the TnOUT registers are incremented at the selected rate (ie. the 8kHz or 64kHz clock source, divided by the selected value).

00FDh - T0OUT - Timer 0 Output (R)
00FEh - T1OUT - Timer 1 Output (R)
00FFh - T2OUT - Timer 2 Output (R)
```
0-3  Incremented at the rate selected via TnDIV (reset to 0 after reading)
4-7  Not used (always zero)
```

SPC700 Waitstates on Internal Cycles
Below lists the number of I/O-Waitstates applied on Internal Cycles of SPC700 opcodes 00h..FFh (that implies: any further Internal Cycles have RAM-Waitstates).
```
00..1F  0,3,0,1,0,0,0,1,0,0,1,0,0,1,0,2, 2,3,0,3,1,1,1,1,0,0,0,1,0,0,0,1
20..3F  0,3,0,1,0,0,0,1,0,0,1,0,0,1,3,2, 0,3,0,3,1,1,1,1,0,0,0,1,0,0,0,3
40..5F  0,3,0,1,0,0,0,1,0,0,0,0,0,1,0,3, 2,3,0,3,1,1,1,1,0,0,0,1,0,0,0,0
60..7F  0,3,0,1,0,0,0,1,0,1,0,0,0,1,2,1, 0,3,0,3,1,1,1,1,1,1,1,1,0,0,0,1
80..9F  0,3,0,1,0,0,0,1,0,0,1,0,0,0,1,0, 2,3,0,3,1,1,1,1,0,0,1,1,0,0,10,3
A0..BF  1,3,0,1,0,0,0,1,0,0,0,0,0,0,1,1, 0,3,0,3,1,1,1,1,0,0,1,1,0,0,1,1
C0..DF  1,3,0,1,0,0,0,1,0,0,1,0,0,0,1,7, 2,3,0,3,1,1,1,1,0,1,0,1,0,0,4,1
E0..FF  0,3,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0, 0,3,0,3,1,1,1,1,0,1,0,1,0,0,3,0
Note: For conditional jumps (with condition=true), add 2 additional cylces.
```
That is, the above list_entries are meant to be used like so:
```
number_of_I/O_waits = list_entry
number_of_RAM_waits = total_number_of_internal_cycles - list_entry
```
For example, Opcode 00h (NOP) has one internal cycle (and it's having RAM timings). Opcode 01h (TCALL) has 3 internal cycles (and all 3 of them have I/O timings).

SNES APU Main CPU Communication Port

2140h - APUI00 - Main CPU to Sound CPU Communication Port 0 (R/W)
2141h - APUI01 - Main CPU to Sound CPU Communication Port 1 (R/W)
2142h - APUI02 - Main CPU to Sound CPU Communication Port 2 (R/W)
2143h - APUI03 - Main CPU to Sound CPU Communication Port 3 (R/W)
```
7-0   APU I/O Data   (Write: Data to APU, Read: Data from APU)
```
Caution: These registers should be written only in 8bit mode (there is a hardware glitch that can cause a 16bit write to [2140h..2141h] to destroy [2143h], this might happen only in some situations, like when the cartridge contains too many ROM chips which apply too much load on the bus).

Uploader
```
Wait until Word[2140h]=BBAAh
kick=CCh                  ;start-code for first command
for block=1..num_blocks
  Word[2142h]=dest_addr   ;usually 200h or higher (above stack and I/O ports)
  Byte[2141h]=01h         ;command=transfer (can be any non-zero value)
  Byte[2140h]=kick        ;start command (CCh on first block)
  Wait until Byte[2140h]=kick
  for index=0 to length-1
    Byte[2141h]=[src_addr+index]      ;send data byte
    Byte[2140h]=index.lsb             ;send index LSB (mark data available)
    Wait until Byte[2140h]=index.lsb  ;wait for acknowledge (see CAUTION)
  next index
  kick=(index+2 AND FFh) OR 1 ;-kick for next command (must be bigger than
next block  ;(if any)         ;         last index+1, and must be non-zero)
[2142h]=entry_point           ;entrypoint, must be below FFC0h (ROM region)
[2141h]=00h                   ;command=entry (must be zero value)
[2140h]=kick                  ;start command
Wait until Byte[2140h]=kick   ;wait for acknowledge
```
CAUTION: The acknowledge for the last data byte lasts only for a few clock cycles, if the uploader is too slow then it may miss it (for example if it's interrupted); as a workaround, disable IRQs and NMIs during upload, or replace the last "wait for ack" by hardcoded delay.

Boot ROM Disassembly
```
FFC0 CD EF        mov  x,EF           ;\
FFC2 BD           mov  sp,x           ; zerofill RAM at [0000h..00EFh]
FFC3 E8 00        mov  a,00           ; (ie. excluding I/O Ports at F0h..FFh)
                 @@zerofill lop:      ; (though [00h..01h] destroyed below)
```

```
FFC5 C6          mov  [x],a          ; (also sets stacktop to 01EFh, kinda
FFC6 1D          dec  x              ; messy, nicer would be stacktop 01FFh)
FFC7 D0 FC       jnz  @@zerofill_lop ;/
FFC9 8F AA F4    mov  [F4],AA        ;\notify Main CPU that APU is ready
FFCC 8F BB F5    mov  [F5],BB        ;/for communication
                 @@wait_for_cc:      ;\
FFCF 78 CC F4    cmp  [F4],CC        ; wait for initial "kick" value
FFD2 D0 FB       jnz  @@wait_for_cc  ;/
FFD4 2F 19       jr   main
                 ;---
                 @@transfer_data:
                 @@wait_for_00:                          ;\
FFD6 EB F4       mov  y,[F4]     ;index (should become 0)    ;
FFD8 D0 FC       jnz  @@wait_for_00                      ;/
                 @@transfer_lop:
FFDA 7E F4       cmp  y,[F4]
FFDC D0 0B       jnz  FFE9          ------->
FFDE E4 F5       mov  a,[F5]     ;get data
FFE0 CB F4       mov  [F4],y     ;ack data
FFE2 D7 00       mov  [[00]+y],a ;store data
FFE4 FC          inc  y          ;addr lsb
FFE5 D0 F3       jnz  @@transfer_lop
FFE7 AB 01       inc  [01]       ;addr msb
                 @@
FFE9 10 EF       jns  @@transfer_lop     ;strange...
FFEB 7E F4       cmp  y,[F4]
FFED 10 EB       jns  @@transfer_lop
                 ;- - -
                 main:
FFEF BA F6       movw ya,[F6]               ;\copy transfer (or entrypoint)
FFF1 DA 00       movw [00],ya    ;addr      ;/address to RAM at [0000h]
FFF3 BA F4       movw ya,[F4]    ;cmd:kick
FFF5 C4 F4       mov  [F4],a     ;ack kick
FFF7 DD          mov  a,y        ;cmd
FFF8 5D          mov  x,a        ;cmd
FFF9 D0 DB       jnz  @@transfer_data
FFFB 1F 00 00    jmp  [0000+x]   ;in: A=0, X=0, Y=0, SP=EFh, PSW=02h
                 ;---
FFFE C0 FF       dw   FFC0  ;reset vector
```

Notes
Many games are jumping to ROM:FFC0h in order to "reset" the SPC700 back to the transmission phase. Some programs are also injumping to ROM:FFC9h (eg. mic_'s "The 700 Club" demo is using that address as dummy-entrypoint after each transfer block; until applying the actual entrypoint after the last block).
Some games may "wrap" from opcodes at RAM:FFBFh to opcodes at ROM:FFC0h. Only known example is World Cup Striker, which contains following code at FFBCh in RAM: "MOV A,80h, MOV [F1h],A", that ensures ROM enabled, and does then continue at FFC0h in ROM (in this specific case, emulators can cheat by handling wraps in their PortF1h handler, rather than checking for PC>=FFC0h after every single opcode fetch).

SNES APU DSP BRR Samples

x4h - VxSRCN - Source number for Voice 0..7 (R/W)
```
0-7   Instrument number (index in DIR table)
```
Points to the BRR Start & Loop addresses (via a table entry at VxSRCN*4+DIR*100h), used when voices are Keyed-ON or Looped.

5Dh - DIR - Sample table address (R/W)
```
0-7   Sample Table Address (in 256-byte steps) (indexed via VxSRCN)
```
The table can contain up to 256 four-byte entries (max 1Kbyte). Each entry is:
```
Byte 0-1  BRR Start Address (used when voice is Keyed-ON)
Byte 2-3  BRR Restart/Loop Address (used when end of BRR data reached)
```
Changing DIR or VxSRCN has no immediate effect (until/unless voices are newly Looped or Keyed-ON).

Bit Rate Reduction (BRR) Format
The sample data consists of 9-byte block(s). The first byte of each block is:
```
7-4  Shift amount   (0=Silent, 12=Loudest, 13-15=Reserved)
3-2  Filter number  (0=None, 1..3=see below)
1-0  Loop/End flags (0..3=see below)
```
The next 8 bytes contain two samples (or nibbles) each:
```
7-4  First Sample  (signed -8..+7)
3-0  Second Sample (signed -8..+7)
```
The Loop/End bits can have following values:
```
Code 0 = Normal   (continue at next 9-byte block)
Code 1 = End+Mute (jump to Loop-address, set ENDx flag, Release, Env=000h)
Code 2 = Ignored  (same as Code 0)
Code 3 = End+Loop (jump to Loop-address, set ENDx flag)
```
The Shift amount is used to convert the 4bit nibbles to 15bit samples:
```
sample = (nibble SHL shift) SAR 1
Accordingly, shift=0 is rather useless (since it strips the low bit).
When shift=13..15, decoding works as if shift=12 and nibble=(nibble SAR 3).
```
The Filter bits allow to select the following filter modes:
```
Filter 0: new = sample
Filter 1: new = sample + old*0.9375
Filter 2: new = sample + old*1.90625  - older*0.9375
Filter 3: new = sample + old*1.796875 - older*0.8125
```
More precisely, the exact formulas are:
```
Filter 0: new = sample
Filter 1: new = sample + old*1+((-old*1) SAR 4)
Filter 2: new = sample + old*2+((-old*3) SAR 5)  - older+((older*1) SAR 4)
Filter 3: new = sample + old*2+((-old*13) SAR 6) - older+((older*3) SAR 4)
```
When creating BRR data, take care that "new" does never exceed -3FFAh..+3FF8h, otherwise a number of hardware glitches will occur:
```
If new>+7FFFh then new=+7FFFh (but, clipped to +3FFFh below) ;\clamp 16bit
If new<-8000h then new=-8000h (but, clipped to ZERO below)   ;/(dirt-effect)
If new=(+4000h..+7FFFh) then new=(-4000h..-1)                ;\clip 15bit
If new=(-8000h..-4001h) then new=(-0..-3FFFh)                ;/(lost-sign)
If new>+3FF8h OR new<-3FFAh then overflows can occur in Gauss section
```
The resulting 15bit "new" value is then passed to the Gauss filter, and additionally re-used for the next 1-2 sample(s) as "older=old, old=new".

BRR Notes
The first 9-byte BRR sample block should always use Filter 0 (so it isn't disturbed by uninitialized old/older values). Same for the first block at the Loop address (unless the old/older values of the initial-pass should happen to match the ending values of the looped-passes).

SNES APU DSP BRR Pitch

x2h - VxPITCHL - Pitch scaler for Voice 0..7, lower 8bit (R/W)
x3h - VxPITCHH - Pitch scaler for Voice 0..7, upper 6bit (R/W)
```
  0-13  Sample rate (0=stop, 3FFFh=fastest) (1000h = 32000Hz)
  14-15 Not used (read/write-able)
```
Defines the BRR sample rate. This register (and PMON) does affect only the BRR sample frequency (but not on the Noise frequency, which is defined - and shared for all voices - in the FLG register).

2Dh - PMON - Pitch Modulation Enable Flags for Voice 1..7 (R/W)
Pitch modulation allows to generate "Frequency Sweep" effects by mis-using the amplitude from channel (x-1) as pitch factor for channel (x).
```
  0    Not used
  1-7  Flags for Voice 1..7 (0=Normal, 1=Modulate by Voice 0..6)
```
For example, output a very loud 1Hz sine-wave on channel 4 (with Direct Gain=40h, and with Left/Right volume=0; unless you actually want to output it to the speaker). Then additionally output a 2kHz sine wave on channel 5 with PMON.Bit5 set. The "2kHz" sound should then repeatedly sweep within 1kHz..3kHz range (or, for a more decent sweep in 1.8kHz..2.2kHz range, drop the Gain level of channel 4).

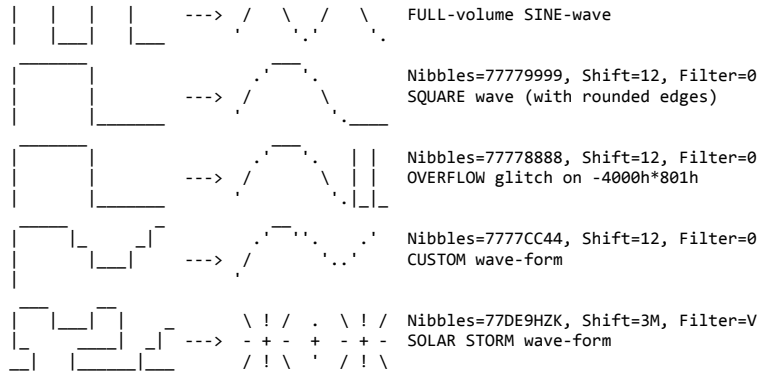
Pitch Counter
The pitch counter is adjusted at 32000Hz rate as follows:
```
  Step = VxPitch                  ;range 0..3FFFh (0..128 kHz)
  IF PMON.Bit(x)=1 AND (x>0)      ;pitch modulation enable
    Factor = VxOUTX(x-1)          ;range -4000h..+3FFFh (prev voice amplitude)
    Factor = (Factor SAR 4)+400h  ;range +000h..+7FFh (factor = 0.00 .. 1.99)
    Step = (Step * Factor) SAR 10 ;range 0..7FEEh (0..256 kHz)
 XXX somewhere here, STEP (or the COUNTER-RESULT) is cropped to 128kHz max) XX?
  Counter = Counter + Step        ;range 0..FFFFh, carry=next BRR block
```
Counter.Bit15-12 indicates the current sample (within a BRR block).
Counter.Bit11-3 are used as gaussian interpolation index.

Maximum Sound Frequency
The pitch counter generates sample rates up to 128kHz. However, the Mixer and DAC are clocked at 32kHz, so the higher rates will skip (or interpolate) samples rather than outputting all samples.
The 32kHz output rate means that one can produce max 16kHz tones.

4-Point Gaussian Interpolation
Interpolation is applied on the 4 most recent 15bit BRR samples (new,old,older,oldest), using bit4-11 of the pitch counter as interpolation index (i=00h..FFh):
```
  out =       ((gauss[0FFh-i] * oldest) SAR 10) ;-initial 16bit value
  out = out + ((gauss[1FFh-i] * older)  SAR 10) ;-no 16bit overflow handling
  out = out + ((gauss[100h+i] * old)    SAR 10) ;-no 16bit overflow handling
  out = out + ((gauss[000h+i] * new)    SAR 10) ;-with 16bit overflow handling
  out = out SAR 1                               ;-convert 16bit result to 15bit
```
The Gauss table contains the following values (in hex):
```
  000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000  ;\
  001,001,001,001,001,001,001,001,001,001,002,002,002,002,002,002  ;
  002,002,003,003,003,003,003,004,004,004,004,004,005,005,005,005  ;
  006,006,006,006,007,007,007,008,008,008,009,009,009,00A,00A,00A  ;
  00B,00B,00B,00C,00C,00D,00D,00E,00E,00F,00F,00F,010,010,011,011  ;
  012,013,013,014,014,015,015,016,017,017,018,018,019,01A,01B,01B  ; entry
  01C,01D,01D,01E,01F,020,020,021,022,023,024,024,025,026,027,028  ; 000h..0FFh
  029,02A,02B,02C,02D,02E,02F,030,031,032,033,034,035,036,037,038  ;
  03A,03B,03C,03D,03E,040,041,042,043,045,046,047,049,04A,04C,04D  ;
  04E,050,051,053,054,056,057,059,05A,05C,05E,05F,061,063,064,066  ;
  068,06A,06B,06D,06F,071,073,075,076,078,07A,07C,07E,080,082,084  ;
  086,089,08B,08D,08F,091,093,096,098,09A,09C,09F,0A1,0A3,0A6,0A8  ;
  0AB,0AD,0AF,0B2,0B4,0B7,0BA,0BC,0BF,0C1,0C4,0C7,0C9,0CC,0CF,0D2  ;
  0D4,0D7,0DA,0DD,0E0,0E3,0E6,0E9,0EC,0EF,0F2,0F5,0F8,0FB,0FE,101  ;
  104,107,10B,10E,111,114,118,11B,11E,122,125,129,12C,130,133,137  ;
  13A,13E,141,145,148,14C,150,153,157,15B,15F,162,166,16A,16E,172  ;/
  176,17A,17D,181,185,189,18D,191,195,19A,19E,1A2,1A6,1AA,1AE,1B2  ;\
  1B7,1BB,1BF,1C3,1C8,1CC,1D0,1D5,1D9,1DD,1E2,1E6,1EB,1EF,1F3,1F8  ;
  1FC,201,205,20A,20F,213,218,21C,221,226,22A,22F,233,238,23D,241  ;
  246,24B,250,254,259,25E,263,267,26C,271,276,27B,280,284,289,28E  ;
  293,298,29D,2A2,2A6,2AB,2B0,2B5,2BA,2BF,2C4,2C9,2CE,2D3,2D8,2DC  ;
  2E1,2E6,2EB,2F0,2F5,2FA,2FF,304,309,30E,313,318,31D,322,326,32B  ; entry
  330,335,33A,33F,344,349,34E,353,357,35C,361,366,36B,370,374,379  ; 100h..1FFh
  37E,383,388,38C,391,396,39B,39F,3A4,3A9,3AD,3B2,3B7,3BB,3C0,3C5  ;
  3C9,3CE,3D2,3D7,3DC,3E0,3E5,3E9,3ED,3F2,3F6,3FB,3FF,403,408,40C  ;
  410,415,419,41D,421,425,42A,42E,432,436,43A,43E,442,446,44A,44E  ;
  452,455,459,45D,461,465,468,46C,470,473,477,47A,47E,481,485,488  ;
  48C,48F,492,496,499,49C,49F,4A2,4A6,4A9,4AC,4AF,4B2,4B5,4B7,4BA  ;
  4BD,4C0,4C3,4C5,4C8,4CB,4CD,4D0,4D2,4D5,4D7,4D9,4DC,4DE,4E0,4E3  ;
  4E5,4E7,4E9,4EB,4ED,4EF,4F1,4F3,4F5,4F6,4F8,4FA,4FB,4FD,4FF,500  ;
  502,503,504,506,507,508,50A,50B,50C,50D,50E,50F,510,511,511,512  ;
  513,514,514,515,516,516,517,517,517,518,518,518,518,518,519,519  ;/
```
The gauss table is slightly bugged: Theoretically, each four values (gauss[000h+i], gauss[0FFh-i], gauss[100h+i], gauss[1FFh-i]) should sum up to 800h, but in practice they do sum up to 7FFh..801h. Of which, 801h can cause math overflows. For example, when outputting three or more "-8 SHL 12" BRR samples with Filter 0, some interpolation results will be +3FF8h (instead of -4000h).
When adding the four new,old,older,oldest values there's some partial overflow handling: The 1st addition can't overflow, the 2nd addition can overflow (when i=0..1Fh), the 3rd addition can overflow (when i=20h..FFh). Of which, the 2nd one bugs, the 3rd one is saturated to Min=-8000h/Max=+7FFFh (giving -4000h/+3FFFh after the final SAR 1).

Waveform Examples
```
  Incoming BRR Data ---> Interpolated Data

  _ _ _ _ _ _
 | | | | | | |         .   .   .   .    Nibbles=79797979, Shift=12, Filter=0
 | | | | | | |   ---> / \ / \ / \ / \   HALF-volume ZIGZAG-wave
 | |_| |_| |_| |_          '   '   '   '
  ___   ___
 |   |_|   |_|           .'.     .'.     Nibbles=77997799, Shift=12, Filter=0
```



SNES APU DSP ADSR/Gain Envelope

x5h - VxADSR1 - ADSR settings for Voice 0..7, lower 8bit (R/W)
x6h - VxADSR2 - ADSR settings for Voice 0..7, upper 8bit (R/W)
```
0-3   4bit Attack rate   ;Rate=N*2+1, Step=+32 (or Step=+1024 when Rate=31)
4-6   3bit Decay rate    ;Rate=N*2+16, Step=-(((Level-1) SAR 8)+1)
7     ADSR/Gain Select   ;0=Use VxGAIN, 1=Use VxADSR (Attack/Decay/Sustain)
8-12  5bit Sustain rate  ;Rate=N, Step=-(((Level-1) SAR 8)+1)
13-15 3bit Sustain level ;Boundary=(N+1)*100h
N/A   0bit Release rate  ;Rate=31, Step=-8 (or Step=-800h when BRR-end)
```
If a voice is Keyed-ON: Enter Attack mode, set Level=0, and increase level. At Level>=7E0h: Switch from Attack to Decay mode (and clip level to max=7FFh if level>=800h). At Level<=Boundary: Switch from Decay to Sustain mode.
If the voice is Keyed-OFF (or the BRR sample contains the Key-Off flag): Switch from Attack/Decay/Sustain/Gain mode to Release mode.

x7h - VxGAIN - Gain settings for Voice 0..7 (R/W)
When VxADSR1.Bit7=1 (Attack/Decay/Sustain Mode):
```
0-7   Not used (instead, Attack/Decay/Sustain parameters are used)
```
When VxADSR1.Bit7=0 and VxGAIN.Bit7=0 (Direct Gain):
```
0-6   Fixed Volume (Envelope Level = N*16, Rate=Infinite)  ;Volume=N*16/800h
7     Must be 0 for this mode
```
When VxADSR1.Bit7=0 and VxGAIN.Bit7=1 (Custom Gain):
```
0-4   Gain rate (Rate=N)
5-6   Gain mode (see below)
7     Must be 1 for this mode
```
In the latter case, the four Gain Modes are:
```
Mode 0 = Linear Decrease  ;Rate=N, Step=-32  (if Level<0 then Level=0)
Mode 1 = Exp Decrease     ;Rate=N, Step=-(((Level-1) SAR 8)+1)
Mode 2 = Linear Increase  ;Rate=N, Step=+32
Mode 3 = Bent Increase    ;Rate=N, If Level<600h then Step=+32 else Step=+8
In all cases, clip E to 0 or 0x7ff rather than wrapping.
```
The Direct/Custom Gain modes are overriding Attack/Decay/Sustain modes (when Keyed-ON), Release (when Keyed-OFF) keeps working as usually.

x8h - VxENVX - Current envelope value for Voice 0..7 (R)
```
0-6  Upper 7bit of the 11bit envelope volume (0..127)
7    Not used (zero)
```
Technically, this register IS writable. But whatever value you write will be overwritten at 32000 Hz.

x9h - VxOUTX - Current sample value for Voice 0..7 (R)
This returns the high byte of the current sample for this voice, after envelope volume adjustment but before VxVOL[LR] is applied.
```
0-7  Upper 8bit of the current 15bit sample value (-128..+127)
```
Technically, this register IS writable. But whatever value you write will be overwritten at 32000 Hz.

ADSR/Gain (and Noise) Rates
The various ADSR/Gain/Noise rates are defined as 5bit rate values (or 3bit/4bit values which are converted to 5bit values as described above). The meaning of these 5bit values is as follows (the table shows the number of 32000Hz sample units it takes until the next Step is applied):
```
00h=Stop   04h=1024  08h=384   0Ch=160   10h=64   14h=24   18h=10   1Ch=4
01h=2048   05h=768   09h=320   0Dh=128   11h=48   15h=20   19h=8    1Dh=3
02h=1536   06h=640   0Ah=256   0Eh=96    12h=40   16h=16   1Ah=6    1Eh=2
03h=1280   07h=512   0Bh=192   0Fh=80    13h=32   17h=12   1Bh=5    1Fh=1
```
Note: All values are "1,3,5 SHL n". Hardware-wise they are probably implemented as 3bit counters, driven at 32kHz, 16kHz, 8kKz, etc. (depending on the shift amount); accordingly, when entering a new ADSR phase, the location of the 1st step may vary depending on when the hardware generates the next 8kHz cycles, for example.

Gain Notes
Even in Gain modes, the hardware does internally keep track of whether it is in Attack/Decay/Sustain phase: In attack phase it switches to Decay at Level>=7E0h. In Decay phase it switches to Sustain at Level<=Boundary (though accidently reading a garbage boundary value from VxGAIN.Bit7-5 instead of from VxADSR2.Bit7-5). Whereas, switching phases doesn't have any audible effect, it just marks the hardware as being in a new phase (and, if software disables Gain-Mode by setting VxADSR1.Bit7, then hardware will process the current phase).
Direct Gain mode can be used to set a fixed volume, or to define the starting point for a different mode (in the latter case, mind that the hardware processes I/O ports only once per sample, so, after setting Direct Gain mode: wait at least 32 cpu cycles before selecting a different mode).

Misc Notes
```
Save the new value, *clipped* to 11 bits, to determine the
increment for GAIN Bent Increase mode next sample. Note that a
negative value for the new value will result in the clipped
version being greater than 0x600.
```

SNES APU DSP Volume Registers

0Ch - MVOLL - Left channel master volume (R/W)
1Ch - MVOLR - Right channel master volume (R/W)

```
  0-7   Volume (-127..+127) (negative = phase inverted) (sample=sample*vol/128)
```
Value -128 causes multiply overflows (-8000h*-80h=-400000h).

x0h - VxVOLL - Left volume for Voice 0..7 (R/W)
x1h - VxVOLR - Right volume for Voice 0..7 (R/W)
```
  0-7   Volume (-128..+127) (negative = phase inverted) (sample=sample*vol/128)
```
Value -128 can be safely used (unlike as for all other volume registers, there is no overflow; because ADSR/Gain does lower the incoming sample to max=sample*7FFh/800h).

Output Mixer
```
  sum =       sample0*V0VOLx SAR 6      ;\
  sum = sum + sample1*V0V1Lx SAR 6      ;
  sum = sum + sample2*V0V2Lx SAR 6      ; with 16bit overflow handling
  sum = sum + sample3*V0V3Lx SAR 6      ; (after each addition)
  sum = sum + sample4*V0V4Lx SAR 6      ;
  sum = sum + sample5*V0V5Lx SAR 6      ;
  sum = sum + sample6*V0V6Lx SAR 6      ;
  sum = sum + sample7*V0V7Lx SAR 6      ;
  sum =       (sum*MVOLx SAR 7)         ;
  sum = sum + (fir_out*EVOLx SAR 7)     ;/
  if FLG.MUTE then sum = 0000h
  sum = sum XOR FFFFh  ;-final phase inversion (as done by built-in post-amp)
```

SNES APU DSP Control Registers

4Ch - KON - Key On Flags for Voice 0..7 (R/W) (W)
```
  0-7  Flags for Voice 0..7 (0=No change, 1=Key On)

       Writing 1 to the KON bit will set the envelope to 0, the state to
       Attack, and will start the channel from the beginning (see DIR and
       VxSRCN). Note that this happens even if the channel is already playing
       (which may cause a click/pop), and that there are 5 'empty' samples
       before envelope updates and BRR decoding actually begin.
```

5Ch - KOFF - Key Off Flags for Voice 0..7 (R/W) (W)
```
  0-7  Flags for Voice 0..7 (0=No change, 1=Key Off)

       Setting 1 to the KOFF bit will transition the voice to the Release
       state. Thus, the envelope will decrease by 8 every sample (regardless
       of the VxADSR and VxGAIN settings) until it reaches 0, where it will
       stay until the next KON.
```

6Ch - FLG - Reset, Mute, Echo-Write flags and Noise Clock (R/W)
The initial value on Reset is E0h (KeyedOff/Muted/EchoWriteOff/NoiseStopped), although reading the FLG register after reset will return a garbage value.
```
  0-4  Noise frequency    (0=Stop, 1=16Hz, 2=21Hz, ..., 1Eh=16kHz, 1Fh=32kHz)
  5    Echo Buffer Writes (0=Enable, 1=Disable) (doesn't disable echo-reads)
  6    Mute Amplifier     (0=Normal, 1=Mute) (doesn't stop internal processing)
  7    Soft Reset         (0=Normal, 1=KeyOff all voices, and set Envelopes=0)
```
Disabling Echo-Writes doesn't affect Echo-Reads (ie. echo buffer is still output, unless Echo Volume L+R or FIR0..7 are set to zero). Mute affects only the external amplifier (internally all sound/echo generation is kept operating).

7Ch - ENDX - Voice End Flags for Voice 0..7 (R) (W=Ack)
```
  0-7  Flags for Voice 0..7 (0=Keyed ON, 1=BRR-End-Bit encountered)
```
Any write to this register will clear ALL bits, no matter what value is written. On reset, all bits are cleared. Though bits may get set shortly after reset or shortly after manual-write (once when the BRR decoder reaches an End-code).
```
       Note that the bit is set at the START of decoding the BRR block, not
       at the end. Recall that BRR processing, and therefore the setting of
       bits in this register, continues even for voices in the Release state.
```

3Dh - NON - Noise Enable Flags for Voice 0..7 (R/W)
Allows to enable Noise on one or more channels, however, there is only one noise-generator (and only one noise frequency) shared for all voices. The 5bit noise frequency is defined in FLG register (see above), and works same as the 5bit ADSR rates (see ADSR chapter for details). The NON bits are:
```
  0-7  Flags for Voice 0..7 (0=Output BRR Samples, 1=Output Noise)
```
The noise generator produces 15bit output level (range -4000h..+3FFFh; initially -4000h after reset). At the selected noise rate, the level is updated as follows:
```
  Level = ((Level SHR 1) AND 3FFFh) OR ((Level.Bit0 XOR Level.Bit1) SHL 14)
```
Caution: Even in noise mode, the hardware keeps decoding BRR sample blocks, and, when hitting a BRR block with End Code 1 (End+Mute), it will switch to Release state with Envelope=0, thus immediately terminating the Noise output. To avoid that, set VxSRCN to an endless looping dummy BRR block.
Note: The VxPITCH registers have no effect on noise frequency, and Gaussian interpolation isn't applied on noise.

KON/KOFF Notes
```
       These registers seem to be polled only at 16000 Hz, when every other
       sample is due to be output. Thus, if you write two values in close
       succession, usually but not always only the second value will have an
       effect:
         ; assume KOFF = 0, but no voices playing
         mov $f2, #$4c  ; KON = 1 then KON = 2
         mov $f3, #$01  ; -> *usually* only voice 2 is keyed on. If both are,
         mov $f3, #$02  ; voice 1 will be *2* samples ahead rather than one.
       and
         ; assume various voices playing
         mov $f2, #$5c  ; KOFF = $ff then KOFF = 0
         mov $f3, #$ff
         mov $f3, #$00  ; -> *usually* all voices remain playing
       FLG bit 7, however, is polled every sample and polled for each voice.

       These registers and FLG bit 7 interact as follows:
         1. If FLG bit 7 or the KOFF bit for the channel is set, transition
            to the Release state. If FLG bit 7 is set, also set the envelope
            to 0.
         2. If the 'internal' value of KON has the channel's bit set, perform
            the KON actions described above.
         3. Set the 'internal' value of KON to 0.
```

```
This has a number of consequences:
  * KON effectively takes effect 'on write', even though a non-zero
    value can be read back much later. KOFF and FLG.7, on the other
    hand, exert their influence constantly until a new value is
    written.
  * Writing KON while KOFF or FLG.7 will not result in any samples
    being output by the channel. The channel is keyed on, but it is
    turned off again 2 samples later. Since there is a 5 sample delay
    after KON before the channel actually beings processing, the net
    effect is no output.
  * However, if KOFF is cleared within 63 SPC700 cycles of the
    KON write above, the channel WILL be keyed on as normal. If KOFF
    is cleared betwen 64 and 127 SPC700 cycles later, the channel
    MIGHT be keyed on with decreasing probability depending on how
    many cycles before the KON/KOFF poll the KON write occurred.
  * Setting both KOFF and KON for a channel will turn the channel
    off much faster than just KOFF alone, since the KON will set the
    envelope to 0. This can cause a click/pop, though.
```

xAh - NA - Unused (8 bytes of general-purpose RAM) (R/W)
xBh - NA - Unused (8 bytes of general-purpose RAM) (R/W)
1Dh - NA - Unused (1 byte of general-purpose RAM) (R/W)
xEh - NA - Unused (8 bytes of general-purpose RAM) (R/W)
These registers seem to have no function at all. Data written to them seems to have no effect on sound output, the written values seem to be left intact (ie. they aren't overwritten by voice or echo status information).

SNES APU DSP Echo Registers

2Ch - EVOLL - Left channel echo volume (R/W)
3Ch - EVOLR - Right channel echo volume (R/W)

```
  0-7   Volume (-128..+127) (negative = phase inverted) (sample=sample*vol/128)
```

This is the adjustment applied to the FIR filter outputs before mixing with the main signal (after master volume adjustment).

0Dh - EFB - Echo feedback volume (R/W)
Specifies the feedback volume (the volume at which the echo-output is mixed back to the echo-input).

```
  0-7   Volume (-128..+127) (negative = phase inverted) (sample=sample*vol/128)
```

Medium values (like 40h) produce "normal" echos (with decreasing volume on each repetition). Value 00h would produce a one-shot echo, value 7Fh would repeat the echo almost infinitely (assuming that FIRx leaves the overall volume unchanged).

4Dh - EON - Echo Enable Flags for Voice 0..7 (R/W)

```
  0-7  Flags for Voice 0..7 (0=Direct Output, 1=Echo (and Direct) Output)
```

When the bit is set and echo buffer write is enabled, this voice will be mixed into the sample to be written to the echo buffer for later echo processing.

6Dh - ESA - Echo ring buffer address (R/W)

```
  0-7   Echo Buffer Base Address (in 256-byte steps)
```

The echo buffer consists of one or more 4-byte entries:

```
  Byte 0: Lower 7bit of Left sample  (stored in bit1-7) (bit0=unused/zero)
  Byte 1: Upper 8bit of Left sample  (stored in bit0-7)
  Byte 2: Lower 7bit of Right sample (stored in bit1-7) (bit0=unused/zero)
  Byte 3: Upper 8bit of Right sample (stored in bit0-7)
```

At 32000Hz rate, the oldest 4-byte entry is removed (and passed on to the FIR filter), and, if echo-writes are enabled in FLG register, the removed entry is then overwritten by new values (from the Echo Mixer; consisting of all Voices enabled in EON, plus echo feedback).
Changes to the ESA register are are realized (almost) immediately: After changing ESA, the hardware may keep access 1-2 samples at [OldBase+index], and does then continue at [NewBase+index] (with index being increased as usually, or reset to 0 when Echo Size has ellapsed).

7Dh - EDL - Echo delay (ring buffer size) (R/W)
Specifies the size of the Echo RAM buffer (and thereby the echo delay). Additionally to that RAM buffer, there is a 8x4-byte buffer in the FIR unit.

```
  0-3  Echo Buffer Size (0=4 bytes, or 1..15=Size in 2K-byte steps) (max=30K)
  4-7  Not used (read/write-able)               (ie. 16 ms steps) (max=240ms)
```

Caution: It may take up to about 240ms until the hardware realizes changes to the EDL register; ie. when re-allocating the buffer size, the hardware may keep writing up to 30Kbytes according to the old size. See Echo Buffer Notes below.

Echo Buffer Notes

```
Note that the ESA register is accessed 32 cycles before the value is
used for a write; at a sample level, this causes writes to appear to be
delayed by at least a full sample before taking effect.

The EDL register value is only used under certain conditions:
 * Write the echo buffer at sample 'idx' (cycles 29 and 30)
 * If idx==0, set idx_max = EDL<<9       (cycle 30-ish)
 * Increment idx. If idx>=idx_max, idx=0 (cycle 30-ish)
This means that it can take up to .25s for a newly written value to
actually take effect, if the old value was 0x0f and the new value is
written just after the cycle 30 in which buffer index 0 was written.
```

xFh - FIRx - Echo FIR filter coefficient 0..7 (R/W)

```
  0-7  Echo Coefficient for 8-tap FIR filter (-80h..+7Fh)
```

Value -128 should not be used for any of the FIRx registers (to avoid multiply overflows). To avoid addition overflows: The sum of POSITIVE values in first seven registers (FIR0..FIR6) should not exceed +7Fh, and the sum of NEGATIVE values should not exceed -7Fh.
The sum of all eight registers (FIR0..FIR7) should be usually around +80h (for leaving the overall output volume unchanged by the FIR unit; instead, echo volumes are usually adjusted via EFB/EVOLx registers).
The FIR formula is:

```
  addr = (ESA*100h+ram_index*4) AND FFFFh           ;-Echo RAM read/write addr
  buf[(i-0) AND 7] = EchoRAM[addr] SAR 1            ;-input 15bit from Echo RAM
  sum =       buf[(i-7) AND 7]*FIR0 SAR 6  ;oldest  ;\
  sum = sum + buf[(i-6) AND 7]*FIR1 SAR 6           ; calculate 16bit sum of
  sum = sum + buf[(i-5) AND 7]*FIR2 SAR 6           ; oldest 7 values, these
  sum = sum + buf[(i-4) AND 7]*FIR3 SAR 6           ; additions are done
  sum = sum + buf[(i-3) AND 7]*FIR4 SAR 6           ; without overflow
  sum = sum + buf[(i-2) AND 7]*FIR5 SAR 6           ; handling
  sum = sum + buf[(i-1) AND 7]*FIR6 SAR 6           ;/
  sum = sum + buf[(i-0) AND 7]*FIR7 SAR 6  ;newest  ;-with overflow handling
```

```
if overflow occurred in LAST addition: saturate to min/max=-8000h/+7FFFh
audio_output=NormalVoices+((sum*EVOLx) SAR 7)     ;-output to speakers
echo_input=EchoVoices+((sum*EFB) SAR 7)           ;-feedback to echo RAM
echo_input=echo_input AND FFFEh                   ;-isolate 15bit/bit0=0
if echo write enabled: EchoRAM[addr]=echo_input   ;-write (if enabled in FLG)
i = i + 1                                         ;-FIR index for next sample
ram_index=ram_index+1                             ;-RAM index for next sample
decrease remain, if remain=0, reload remain from EDL and set ram_index=0
```

Note that the left and right stereo channels are filtered separately (no crosstalk), but with identical coefficients.

Filter Examples

```
FIR0 FIR1 FIR2 FIR3 FIR4 FIR5 FIR6 FIR7  EFB
FF   08   17   24   24   17   08   FF    40  Echo with Low-pass (bugged)
7F   00   00   00   00   00   00   00    7F  Echo (nearly endlessly repeated)
7F   00   00   00   00   00   00   00    40  Echo (repeat w. decreasing vol)
7F   00   00   00   00   00   00   00    00  Echo (one-shot)
```

Note: The "bugged" example (with the sum of FIR0..FIR6 exceeding +7Fh) is from official (!) specs (and thus possibly actually used by some games(?)).

Echo Overflows

Setting FIRx, EFB, or EVOLx to -128 does probably cause multiply overflows?

SNES APU Low Level Timings

Register and RAM Access Timing Chart

On every CPU cycle, the hardware can access 3 bytes from RAM (2 DSP accesses, and 1 CPU access), and 4 bytes from DSP Register Array (3 DSP accesses, and 1 CPU access), plus some special DSP Registers (ENDX/FLG). Audio is generated at 32000Hz Rate (every 32 CPU cycles):

```
Time <--- RAM Access --->   <---- Register Array ---->   <--Extra-->
T0   [V0BRR.2nd.dta1]       --      ,V0VOLL    ,V2SRCN
T1   [V1SRCN/DIR.lsb/msb]   V0VOLR  ,V1PITCHL  ,V1ADSR1  ENDX.0, Timer0,1,2
T2   [V1BRR.1st.hdr/dta0]   V0ENVX  ,V1PITCHH  ,V1ADSR2  V1:FLG.7
T3   [V1BRR.2nd.dta1]       V0OUTX  ,V1VOLL    ,V3SRCN
T4   [V2SRCN/DIR.lsb/msb]   V1VOLR  ,V2PITCHL  ,V2ADSR1  ENDX.1
T5   [V2BRR.1st.hdr/dta0]   V1ENVX  ,V2PITCHH  ,V2ADSR2  V2:FLG.7
T6   [V2BRR.2nd.dta1]       V1OUTX  ,V2VOLL    ,V4SRCN
T7   [V3SRCN/DIR.lsb/msb]   V2VOLR  ,V3PITCHL  ,V3ADSR1  ENDX.2
T8   [V3BRR.1st.hdr/dta0]   V2ENVX  ,V3PITCHH  ,V3ADSR2  V3:FLG.7
T9   [V3BRR.2nd.dta1]       V2OUTX  ,V3VOLL    ,V5SRCN
T10  [V4SRCN/DIR.lsb/msb]   V3VOLR  ,V4PITCHL  ,V4ADSR1  ENDX.3
T11  [V4BRR.1st.hdr/dta0]   V3ENVX  ,V4PITCHH  ,V4ADSR2  V4:FLG.7
T12  [V4BRR.2nd.dta1]       V3OUTX  ,V4VOLL    ,V6SRCN
T13  [V5SRCN/DIR.lsb/msb]   V4VOLR  ,V5PITCHL  ,V5ADSR1  ENDX.4
T14  [V5BRR.1st.hdr/dta0]   V4ENVX  ,V5PITCHH  ,V5ADSR2  V5:FLG.7
T15  [V5BRR.2nd.dta1]       V4OUTX  ,V5VOLL    ,V7SRCN
T16  [V6SRCN/DIR.lsb/msb]   V5VOLR  ,V6PITCHL  ,V6ADSR1  ENDX.5
T17  [V6BRR.1st.hdr/dta0]   V5ENVX  ,V6PITCHH  ,V6ADSR2  V6:FLG.7, Timer2
T18  [V6BRR.2nd.dta1]       V5OUTX  ,V6VOLL    ,V0SRCN
T19  [V7SRCN/DIR.lsb/msb]   V6VOLR  ,V7PITCHL  ,V7ADSR1  ENDX.6
T20  [V7BRR.1st.hdr/dta0]   V6ENVX  ,V7PITCHH  ,V7ADSR2  V7:FLG.7
T21  [V7BRR.2nd.dta1]       V6OUTX  ,V7VOLL    ,V1SRCN
T22  [V0SRCN/DIR.lsb/msb]   V7VOLR  ,V0PITCHL  ,V0ADSR1  ENDX.7
T23  [RdEchoLeft.lsb/msb]   V7ENVX  ,V0PITCHH  ,FIR0
T24  [RdEchoRight.lsb/msb]  V7OUTX  ,FIR1      ,FIR2
T25  --- ;SRAM./OE=no       FIR3    ,FIR4      ,FIR5
T26  [V0BRR.1st.hdr/dta0]   FIR6    ,FIR7      ,---
T27  --- ;SRAM./OE=yes?!    MVOLL   ,EVOLL     ,EFB
T28  --- ;SRAM./OE=yes?!    MVOLR   ,EVOLR     ,PMON
T29  --- ;SRAM./OE=no       NON     ,EON       ,DIR      FLG.5
T30  [WrEchoLeft.lsb/msb]   EDL     ,ESA       ,KON?     FLG.5
T31  [WrEchoRight.lsb/msb]  KOFF    ,FLG.LSB   ,V0ADSR2  FLG.0-4,V0:FLG.7,KON
```

Note: In Gain-Mode, above reads VxGAIN instead of VxADSR2.
KON and KOFF are processed only each 64 clk cycles (each 2nd pass).
The SPC and DSP chips are started via same /RESET and clocked via same 2.048MHz signal, causing the SPC Timers to be incremented in sync with DSP timings: at T1 and T17 (unless one pauses the SPC Timers via TEST register).

Additionally (non-RAM-array):

```
ENDX         8bits
KON-changed  1bit
FLG.MSBs     3bit (or maybe/rather it's full 8bit, including LSBs?)
```

Internal Signals

LRCK is HIGH during T0..T15, and LOW during T16..T31.

APU Signals

```
Time  15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31  0  1  (Tnn)
LRCK  ---_______________________________________________------ (DSP.Pin43)
MX1   _-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__-- (DSP.Pin3)
MX2   __-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__-__- (DSP.Pin4)
MX3   __'__'__'__'__'__'__'__'__'__'__'__'__'__'__'__'__'__' (DSP.Pin5)
/WE   --C--C--C--C--C--C--C--C--C--C--C--C--C--C--CEECEEC--C--C (SRAM.Pin27)
/OE   DDCDDCDDCDDCDDCDDCDDCDDCEECEECddCDDCddCddCddC--C--CDDCDDC (SRAM.Pin22)
/CE0  DDCDDCDDCDDCDDCDDCDDCDDCDDC--C--C--CDDCddCddC--C--C--CDDCDDC (SRAM.Pin20a)
/CE1  --C--C--C--C--C--C--C--C--CEECEEC--C--C--C--C--CEECEEC--C--C (SRAM.Pin20b)
```

Whereas:

```
"-"  High
"_"  Low
"'"  Very short High (near falling edges of MX2)
"C"  CPU access           ;-(occurs when MX2=High)
"EE" Echo access          ;\
"DD" DSP access (DIR/BRR) ; (occurs when MX2=Low)
"dd" DSP access (dummy)   ;/
```

For /CE0 and /CE1: Assume DIR/BRR in lower 32K, Echo in upper 32K RAM.
The "DD" and "dd" signals seem to be always going Low (even when no new BRR/DIR data is needed).

SNES Maths Multiply/Divide

4202h - WRMPYA - Set unsigned 8bit Multiplicand (W)
4203h - WRMPYB - Set unsigned 8bit Multiplier and Start Multiplication (W)
Set WRMPYA (or leave it unchanged, if it already contains the desired value), then set WRMPYB, wait 8 clk cycles, then read the 16bit result from Port 4216h-4217h. For some reason, the hardware does additionally set RDDIVL=WRMPYB, and RDDIVH=00h.

4204h - WRDIVL - Set unsigned 16bit Dividend (lower 8bit) (W)
4205h - WRDIVH - Set unsigned 16bit Dividend (upper 8bit) (W)
4206h - WRDIVB - Set unsigned 8bit Divisor and Start Division (W)
Set WRDIVL/WRDIVH (or leave it unchanged, if it already contains the desired value), then set WRDIVB, wait 16 clk cycles, then read the 16bit result and/or 16bit remainder from Port 4214h-4217h.
Division by zero returns Result=FFFFh, Remainder=Dividend. Note: Almost all commercial SNES games are zero-filling I/O ports upon initialization, thereby causing division by zero (so, debuggers should ignore division errors).

4214h - RDDIVL - Unsigned Division Result (Quotient) (lower 8bit) (R)
4215h - RDDIVH - Unsigned Division Result (Quotient) (upper 8bit) (R)
See Ports 4204h-4206h (divide). Destroyed by 4203h (multiply).

4216h - RDMPYL - Unsigned Division Remainder / Multiply Product (lo.8bit) (R)
4217h - RDMPYH - Unsigned Division Remainder / Multiply Product (up.8bit) (R)
See Ports 4204h-4206h (divide), and 4202h-4203h (multiply).

Timing Notes
The 42xxh Ports are clocked by the CPU Clock, meaning that one needs the same amount of "wait" opcodes no matter if the CPU Clock is 3.5MHz or 2.6MHz. When reading the result, the "MOV r,[421xh]" opcode does include 3 cycles (spent on reading the 3-byte opcode), meaning that one needs to insert only 5 cycles for MUL and only 13 for DIV.
Some special cases: If the the upper "N" bits of 4202h are all zero, then it seems that one may wait "N" cycles less. If memory REFRESH occurs (once and when), then the result seems to be valid within even less wait opcodes.
The maths operations are started only on WRMPYB/WRDIVB writes (not on WRMPYA/WRDIVL/WRDIVH writes; unlike the PPU maths which start on any M7A/M7B write).

=== PPU Ports ===

Below Ports 21xxh are PPU registers. The registers are also used for rotation/scaling effects in BG Mode 7. In BG Mode 0-6 they can be used freely for multiplications. In Mode 7 they are usable ONLY during V-Blank and Forced-Blank (during the Mode 7 Drawing & H-Blank periods, they return garbage in MPYL/MPYM/MPYH, and of course writing math-parameters to M7A/M7B would also mess-up the display).

211Bh - M7A - Rotation/Scaling Parameter A (and Maths 16bit operand) (W)
```
  1st Write: Lower 8bit of signed 16bit Multiplicand  ;\1st/2nd write mechanism
  2nd Write: Upper 8bit of signed 16bit Multiplicand  ;/uses "M7_old" (Mode7)
```

211Ch - M7B - Rotation/Scaling Parameter B (and Maths 8bit operand) (W)
```
  Any Write: Signed 8bit Multiplier                   ;-also affects "M7_old"
```
After writing to 211Bh or 211Ch, the result can be read immediately from 2134h-2136h (the 21xxh Ports are rapidly clocked by the PPU, there's no delay needed when reading via "MOV A,[211Ch]" or via "MOV A,[1Ch]" (with D=2100h), both works even when the CPU runs at 3.5MHz).

2134h - MPYL - Signed Multiply Result (lower 8bit) (R)
2135h - MPYM - Signed Multiply Result (middle 8bit) (R)
2136h - MPYH - Signed Multiply Result (upper 8bit) (R)
See Ports 211Bh-211Ch.

Notes
Some cartridges contain co-processors with further math functions:
SNES Cartridges

SNES Controllers

I/O Ports
SNES Controllers I/O Ports - Automatic Reading
SNES Controllers I/O Ports - Manual Reading

Controller IDs
SNES Controllers Hardware ID Codes
SNES Controllers Detecting Controller Support of ROM-Images

Standard Controllers
SNES Controllers Joypad
SNES Controllers Mouse (Two-button Mouse)
SNES Controllers Multiplayer 5 (MP5) (Five Player Adaptor)

Light Guns
SNES Controllers SuperScope (Lightgun)
SNES Controllers Konami Justifier (Lightgun)
SNES Controllers M.A.C.S. (Lightgun)

Other controllers
SNES Controllers Twin Tap
SNES Controllers Miracle Piano
SNES Controllers NTT Data Pad (joypad with numeric keypad)
SNES Controllers X-Band Keyboard
SNES Controllers Tilt/Motion Sensors
SNES Controllers Lasabirdie (golf club)
SNES Controllers Exertainment (bicycle exercising machine)

SNES Controllers Pachinko
SNES Controllers Other Inputs

Other devices that connect to the controller port
SNES Add-On Turbo File (external backup memory for storing game positions)
SNES Add-On Barcode Battler (barcode reader)
SNES Add-On SFC Modem (for JRA PAT)
SNES Add-On Voice-Kun (IR-transmitter/receiver for use with CD Players)

Pinouts
SNES Controllers Pinouts

SNES Controllers I/O Ports - Automatic Reading

4218h/4219h - JOY1L/JOY1H - Joypad 1 (gameport 1, pin 4) (R)
421Ah/421Bh - JOY2L/JOY2H - Joypad 2 (gameport 2, pin 4) (R)
421Ch/421Dh - JOY3L/JOY3H - Joypad 3 (gameport 1, pin 5) (R)
421Eh/421Fh - JOY4L/JOY4H - Joypad 4 (gameport 2, pin 5) (R)

Register Bit Number	Serial Transfer Order	Default Purpose (Joypads)	
15	1st	Button B	(1=Low=Pressed)
14	2nd	Button Y	
13	3rd	Select Button	
12	4th	Start Button	
11	5th	DPAD Up	
10	6th	DPAD Down	
9	7th	DPAD Left	
8	8th	DPAD Right	
7	9th	Button A	
6	10th	Button X	
5	11th	Button L	
4	12th	Button R	
3	13th	0 (High)	
2	14th	0 (High)	
1	15th	0 (High)	
0	16th	0 (High)	

Before reading above ports, set Bit 0 in port 4200h to request automatic reading, then wait until Bit 0 of port 4212h gets set-or-cleared? Once 4200h enabled, seems to be automatically read on every retrace?
Be sure that Out0 in Port 4016h is zero (otherwise the shift register gets stuck on the first bit, ie. all 16bit will be equal to the B-button state.

```
AUTO JOYPAD READ
----------------
When enabled, the SNES will read 16 bits from each of the 4 controller port
data lines into registers $4218-f. This begins between H=32.5 and H=95.5 of
the first V-Blank scanline, and ends 4224 master cycles later. Register $4212
bit 0 is set during this time. Specifically, it begins at H=74.5 on the first
frame, and thereafter some multiple of 256 cycles after the start of the
previous read that falls within the observed range.

Reading $4218-f during this time will read back incorrect values. The only
reliable value is that no buttons pressed will return 0 (however, if buttons
are pressed 0 could still be returned incorrectly). Presumably reading $4016/7
or writing $4016 during this time will also screw things up.
```

SNES Controllers I/O Ports - Manual Reading

4016h/Write - JOYWR - Joypad Output (W)

```
7-3  Not used
2    OUT2, Output on CPU Pin 39 (seems to be not connected) (1=High)
1    OUT1, Output on CPU Pin 38 (seems to be not connected) (1=High)
0    OUT0, Output on CPU Pin 37 (Joypad Strobe) (both gameports, pin 3)
```

Out0-2 are found on CPU Pins 37-39, of which only Out0 seems to be connected.
Note: The NSS (arcade cabinet) uses OUT2 to signalize Game Over to the Z80 coprocessor.

4016h/Read - JOYA - Joypad Input Register A (R)

```
7-2  Not used
1    Input on CPU Pin 33, connected to gameport 1, pin 5 (JOY3) (1=Low)
0    Input on CPU Pin 32, connected to gameport 1, pin 4 (JOY1) (1=Low)
```

Reading from this register automatically generates a clock pulse on CPU Pin 35, which is connected to gameport 1, pin 2.

4017h/Read - JOYB - Joypad Input Register B (R)

```
7-5  Not used
4    Input on CPU Pin 31, connected to GND (always 1=LOW)       (1=Low)
3    Input on CPU Pin 30, connected to GND (always 1=LOW)       (1=Low)
2    Input on CPU Pin 29, connected to GND (always 1=LOW)       (1=Low)
1    Input on CPU Pin 28, connected to gameport 2, pin 5 (JOY4) (1=Low)
0    Input on CPU Pin 27, connected to gameport 2, pin 4 (JOY2) (1=Low)
```

Reading from this register automatically generates a clock pulse on CPU Pin 36, which is connected to gameport 2, pin 2.

4201h - WRIO - Joypad Programmable I/O Port (Open-Collector Output) (W)

```
7-0   I/O PORT  (0=Output Low, 1=HighZ/Input)
7     Joypad 2 Pin 6 / PPU Lightgun input (should be usually always 1=Input)
6     Joypad 1 Pin 6
5-0   Not connected (except, used by SFC-Box; see Hotel Boxes)
```

Note: Due to the weak high-level, the raising "edge" is raising rather slowly, for sharper transitions one may need external pull-up resistors.

4213h - RDIO - Joypad Programmable I/O Port (Input) (R)

```
7-0   I/O PORT  (0=Low, 1=High)
```

When used as Input via 4213h, set the corresponding bits in 4201h to HighZ.

I/O Signals 0..7 are found on CPU Pins 19..26 (in that order). IO-6 connects to Pin 6 of Controller 1. IO-7 connects to Pin 6 of Controller 2, this pin is also shared for the light pen strobe.
Wires are connected to IO-0..5, but the wires disappear somewhere in the multi-layer board (and might be dead-ends), none of them is output to any connectors (not to the Controller ports, not to the cartridge slot, and not to the EXT port).

SNES Controllers Hardware ID Codes

Controller ID Bits (13th-16th bit, or extended: 13th-24th bit)

```
13th ... 24th Hex  Type
0000.00000000 0.00 No controller connected
0000.11111111 0.FF Normal Joypad (probably also 3rd-party joypads/joysticks)
0001          1    Mouse
0010          2 ?  Unknown (if any)
0011          3    SFC Modem (used by JRA PAT)
0100          4    NTT Data Controller Pad (used by JRA PAT)
....          5-C  Unknown (if any)
1101          D    Voice-Kun (IR-transmitter/receiver, for CD Players)
1110.xxxxxxxx E.xx Third-Party Devices (see below)
1110.000000xx E.0x Epoch Barcode Battler II (detection requires DELAYS?!)
1110.01010101 E.55 Konami Justifier
1110.01110111 E.77 Sunsoft Pachinko Controller
1110.11111110 E.FE ASCII Turbo File Twin in STF mode
1110.11111111 E.FF ASCII Turbo File Twin in TFII mode (or Turbo File Adapter)
1111          F    Nintendo Super Scope
N/A           N/A  M.A.C.S. (no ID, returns all bits = trigger button)
```

Note: The Multiplayer 5 can be also detected (using a different mechansim than reading bits 13th-16th).

Devices with unknown IDs (if they do have any special controller IDs at all)

```
Lasabirdie            ;\
Twin Tap              ; these should have custom IDs?
Miracle Piano         ;
X-Band Keyboard       ;/
Exertainment          ;-connects to expansion port (thus no controller id)
BatterUP                                          ;\
TeeV Golf                                         ; these might return
StuntMaster                                       ; normal "joypad" ID?
Nordic Quest                                      ;
Hori SGB Commander (in normal mode / in SGB mode) ;
Nintendo Joysticks                                ;/
```

SNES Controllers Detecting Controller Support of ROM-Images

Below are some methods to detect controller support by examining ROM-images. The methods aren't fail-proof, but may be useful to track-down controller support in many games.

Detection Method Summary

```
Type              Method
Joypad            <none/default>
Mouse             String "START OF MOUSE BIOS", or opcodes (see below)
Multiplayer 5     String "START OF MULTI5 BIOS"
Super Scope       String "START OF SCOPE BIOS", or Title=<see list>
Lasabirdie        String "GOLF_READY!"
X-Band Keyboard   String "ZSAW@",x,x,"CXDE$#" (keyboard translation table)
Turbo File (STF)  String "FAT0SHVC"
Turbo File (TFII) Opcodes "MOV Y,000Fh/MOV A,[004017h]/DEC Y/JNZ $-5"
Exertainment      Opcodes "MOV [21C1h],A/MOV A,0Bh/MOV [21C4h],A/MOV X,20F3h"
Barcode Battler   Opcodes "INC X/CMP X,(00)0Ah/JNC $-6(-1)/RET/36xNOP/RET"
Voice-Kun         Opcodes "MOV [004201h],A/CLR P,20h/MOV A,D/INC A"
Justifier         Title="LETHAL ENFORCERS     "
M.A.C.S.          Title="MAC:Basic Rifle      "
Twin Tap          Title="QUIZ OH SUPER        "
Miracle Piano     Title="MIRACLE              "
NTT Data Pad      Title="NTT JRA PAT          "
SFC Modem         Title="NTT JRA PAT          "
Pachinko          Title=CB,AF,BB,C2,CA,DF,C1,DD,CB,AF,BB,C2,CA,DF,C1,DD,xx(6)
BatterUP          -  ;\
TeeV Golf         -  ; these are probably simulating standard joypads
StuntMaster       -  ; (and thus need no detection)
Nordic Quest      -  ;/
```

START OF xxx BIOS Strings
These strings were included in Nintendo's hardware driver source code files, the strings themselves have no function (so one could simply remove them), but Nintendo prompted developers to keep them included. They are typically arranged like so:

```
"START OF xxx BIOS"
[bios program code...]
"NINTENDO SHVC xxx BIOS VER x.xx"
"END OF xxx BIOS"
```

Whereas, the version string may be preceeded by "MODIFIED FROM ", and "VER" may be "Ver" in some cases, sometimes without space between "Ver" and "x.xx". In case of custom code one may omit the version string or replace it by "MY BIOS VERSION" or so, but one should include the "START/END OF xxx BIOS" strings to ease detection.

Multiplayer 5
Games that do SUPPORT the hardware should contain following string:

```
"START OF MULTI5 BIOS"
```

Games that do DETECT the hardware should contain following string:

```
"START OF MULTI5 CONNECT CHECK"
```

Games that contain only the "CHECK" part (but not the "BIOS" part) may REJECT to operate with the hardware.
Some MP5 games (eg. "Battle Cross") do lack the "START OF ..." strings.

Mouse
Games that do SUPPORT the hardware should contain following string:

```
"START OF MOUSE BIOS"
```

Some Mouse games (eg. Arkanoid Doh It Again) do lack the "START OF ..." strings. For such games, checking following opcodes may help:

```
MOV Y,0Ah/LOP:/MOV A,[(00)4016h+X]/DEC Y/JNZ LOP/MOV A,[(00)4016h+X]
```

The official mouse BIOS uses 24bit "004016h+X" (BF 16 40 00), Arkanoid uses 16bit "4016h+X" (BD 16 40).
Warning: Automatically activiting mouse-emulation for mouse-compatible games isn't a very good idea: Some games expect the mouse in port 1, others in port 2, so there's a 50% chance to pick the wrong port. Moreover, many games are deactivating normal joypad input when sensing a connected mouse, so automatic mouse emulation will also cause automatic problems with normal joypad input.

SuperScope
Games that do SUPPORT the hardware should (but usually don't) contain following string:

```
"START OF SCOPE BIOS"
```

In practice, the string is included only in "Yoshi's Safari" whilst all other (older and newer) games contain entirely custom differently implemented program code without ID strings, making it more or less impossible to detect SuperScope support. One workaround would be to check known Title strings:

```
"BATTLE CLASH          "    "METAL COMBAT         "    "T2 ARCADE            "
"BAZOOKA BLITZKRIEG    "    "OPERATION THUNDERBOLT"    "TINSTAR              "
"Hunt for Red October "     "SPACE BAZOOKA        "    "X ZONE               "
"LAMBORGHINI AMERICAN "     "SUPER SCOPE 6        "    "YOSHI'S SAFARI       "
"Lemmings 2,The Tribes"
```

Lasabirdie
Games that do support the hardware should contain "GOLF_READY!" string (used to verify the ID received from the hardware).

Turbo File Twin in STF Mode
Games that do support the hardware should contain "FAT0" and SHVC" strings, which are usually (maybe always) stored as continous "FAT0SHVC" string.

Turbo File Twin in TFII Mode or Turbo File Adapter
There aren't any specific ASCII Strings. However, most (or all) games contain the "MOV Y,000F, MOV A,[004017], DEC Y, JNZ $-5" opcode sequence (exactly like so, ie. with Y=16bit, and address=24bit).

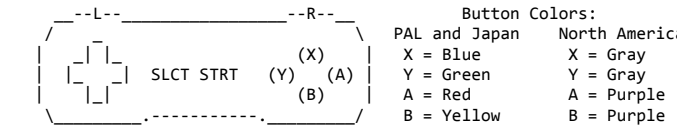
NSRT Header
Some ROM-images do contain information about supported controllers in NSRT Headers. In practice, most ROM-images don't have that header (but can be "upgraded" by Nach's NSRT tool).
SNES Cartridge ROM-Image Headers and File Extensions
The NSRT format isn't officially documented. The official way to create headers seems to be to contact the author (Nach), ask him to add controller flags for a specific game, download the updated version of the NSRT tool, use it to update your ROM-image, and then you have the header (which consists of undocumented, and thereby rather useless values).

SNES Controllers Joypad

Joypad Bits

```
1st          Button B        (0=High=Released, 1=Low=Pressed)
2nd          Button Y        (0=High=Released, 1=Low=Pressed)
3rd          Button Select   (0=High=Released, 1=Low=Pressed)
4th          Button Start    (0=High=Released, 1=Low=Pressed)
5th          Direction Up    (0=High=Released, 1=Low=Pressed)
6th          Direction Down  (0=High=Released, 1=Low=Pressed)
7th          Direction Left  (0=High=Released, 1=Low=Pressed)
8th          Direction Right (0=High=Released, 1=Low=Pressed)
9th          Button A        (0=High=Released, 1=Low=Pressed)
10th         Button X        (0=High=Released, 1=Low=Pressed)
11th         Button L        (0=High=Released, 1=Low=Pressed)
12th         Button R        (0=High=Released, 1=Low=Pressed)
13th         ID Bit3         (always 0=High)
14th         ID Bit2         (always 0=High, except 1=Low for NTT Data Pad)
15th         ID Bit1         (always 0=High)
16th         ID Bit0         (always 0=High)
17th and up  Padding         (always 1=Low) (or 0=High when no pad connected)
```

Joypad Physical Appearance



Joypad/Joystick Variants
There are numerous variants from various companies, some including extra features like auto-fire.

```
Advanced Control Pad (Mad Catz) (joypad with autofire or so)
Angler (?) Functionally identical to the ASCII Pad (optional "stick" in dpad)
asciiGrip (ASCII) (normal joypad for single-handed use)
asciiPad (ASCIIWARE) (joypad with autofire and slowmotion)
Capcom Pad Soldier (Capcom) (standard pad in bent/squeezed/melted design)
Competition Pro (Competition Pro) (joypad with autofire and slowmotion)
Competition Pro (Competition Pro) (slightly redesigned standard joypad)
Conqueror 2 (QuickShot?) (joystick with autofire, programmable buttons)
Cyberpad (Quickshot?) (6-shaped pad, programmable, autofile, slow motion)
Dual Turbo (Akklaim) (set of 2 wireless joypads with autofire or so)
Energiser (?) (very odd shaped pad, programmable, auto fire, slow motion)
Fighter Stick SN (?) (desktop joystick, with autofire or so)
Gamemaster (Triton) (edgy-shaped pad, one programmable button)
High Frequency Control Pad (High Frequency) (normal pad, wrong button colors)
Invader 2 (QuickShot?) (joypad with autofire)
JS-306 Power Pad Tilt (Champ) (joypad with autofire, slowmotion, tilt-mode)
Multisystem 6 (Competition Pro) (pad supports Genesis and SNES)
Nigal Mouncefill Fly Wheel (Logic 3) (wheel-shaped, tilt-sensor instead dpad)
NTT Data Pad (for JRA PAT) (joypad with numeric keypad) (special ID)
Pro Control 6 (Naki) (joypad, programmable & whatever extra features)
Pro-Player (?) (joystick)
Score Master (Nintendo) (desktop joystick with autofire or so)
SF-3 (Honey Bee) (very flat normal pad with autofire)
SGB Controller (?) (joypad ...)
```

```
SN Propad
SN Propad 2
SN Propad 6
SN-6 (Gamester) (standard joypad clone)
Specialized Fighter Pad (ASCIIWARE) (autofire, L/R as "normal" buttons)
Speedpad (?) (joypad, one auto-switch, L/R buttons as "normal" buttons)
Super Control Pad (?) (standard joypad clone, plus 3-position switch?)
Super Joy Card (Hudson) (standard joypad with auto-fire or so)
Supercon (QuickShot) (standard joypad, odd shape, odd start/select buttons)
Superpad (InterAct) (standard joypad clone)
Superpad (noname) (standard joypad)
TopFighter (?) (desktop joystick, programmable, LCD panel, auto-fire, slowmo)
Turbo Touch 360 (Triax) (joypad with autofire)
V356 (Recoton) (normal joypad, with whatever 3-position switch)
noname joypads (normal joypad clones without nintendo text nor snes logo)
joypad (Konami) (wireless joypad, no extra functions) (dish-shaped receiver)
joypads (Game Partner) (set of 2 wireless joypads with autofire or so)
AK7017828 or so??? (Game Partner) (joypad, slow motion, auto fire)
Noname pad (Tomee) (standard joypad clone)
SNES+MD? (Nakitek) (joypad with whatever special features)
```

Capcom Fighter Power Stick
Super Advantage Joystick
SGB Commander (HORI)
Battle Pachislot Controller (Sammy) (joypad for "one-armed bandit" games)

Power Plug (Tyco)
Auto-fire adaptor, plugs between any joypad/joystick and snes console.

SNES Controllers Mouse (Two-button Mouse)

Mouse Connection
The mouse can be connected to Controller Port 1 or 2. Default seems to be Port 1 for most games. Exception: Satellaview FLASH games should default to Port 2 (the joypad controlled BS-X BIOS doesn't work with mouse plugged into Port 1).
Mario Paint accepts it ONLY in Port 1, other games may accept either port (maybe there are also some that accept only Port 2?). Two-player games (eg. Operation Thunderbolt) may accept two mice to be connected. Some games (eg. Super Bomberman Panic Bomber World) refuse to run if a mouse is connected. The mouse should not be connected to Multiplayer 5 adaptors (which allow only 17mA per controller, whilst the mouse requires 50mA).

Supported Games
SNES Controllers Mouse Games

Mouse Bits

```
1st..8th     Unused       (always 0=High)
9th          Right Button (0=High=Released, 1=Low=Pressed)
10th         Left Button  (0=High=Released, 1=Low=Pressed)
11th         Sensitivity Bit1   (0=High=Zero)     ;\0=slow, 1=normal, 2=fast
12th         Sensitivity Bit0   (0=High=Zero)     ;/
13th         ID Bit3      (always 0=High)
14th         ID Bit2      (always 0=High)
15th         ID Bit1      (always 0=High)
16th         ID Bit0      (always 1=Low)
17th         Vertical Direction     (0=High=Down, 1=Low=Up)
18th         Vertical Offset Bit6   (0=High=Zero)    ;\
19th         Vertical Offset Bit5   (0=High=Zero)    ;
20th         Vertical Offset Bit4   (0=High=Zero)    ; this is a 7bit
21th         Vertical Offset Bit3   (0=High=Zero)    ; UNSIGNED value
22th         Vertical Offset Bit2   (0=High=Zero)    ; (00h=No motion)
23th         Vertical Offset Bit1   (0=High=Zero)    ;
24th         Vertical Offset Bit0   (0=High=Zero)    ;/
25th         Horizontal Direction   (0=High=Right, 1=Low=Left)
26th         Horizontal Offset Bit6 (0=High=Zero)    ;\
27th         Horizontal Offset Bit5 (0=High=Zero)    ;
28th         Horizontal Offset Bit4 (0=High=Zero)    ; this is a 7bit
29th         Horizontal Offset Bit3 (0=High=Zero)    ; UNSIGNED value
30th         Horizontal Offset Bit2 (0=High=Zero)    ; (00h=No motion)
31th         Horizontal Offset Bit1 (0=High=Zero)    ;
32th         Horizontal Offset Bit0 (0=High=Zero)    ;/
33th and up  Padding      (always 1=Low)
```

Note that the motion values consist of a Direction Bit and an UNSIGNED 7bit offset (ie. not a signed 8bit value). After reading, the 7bit offsets are automatically reset to zero (whilst the direction bits do reportedly stay unchanged unless/until the mouse is moved in opposite direction).

Mouse Support ID-String
Games that support the mouse should contain the string "START OF MOUSE BIOS" somewhere in the ROM-image.

Mouse Sensitivity
The Mouse Resolution is specified as "50 counts/inch (+/-10%)". There are three selectable Sensitivity (Threshold) settings:

```
0 - slow   - linear fixed level (1:1)
1 - normal - exponential -?- levels (1:1 to ?:1)  (?:1=smaller than 6:1)
2 - fast   - exponential six levels (1:1 to 6:1)
```

Setting 0 returns raw mickeys (so one must implement effects like double-speed threshold by software). Settings 1-2 can be used directly as screen-pixel offsets.
To change the sensitivity (for port n=0 or n=1):

```
[4016h]=01h           ;set STB=1
dummy=[4016h+n]       ;issue CLK pulse while STB=1 <-- increments the value,
[4016h]=00h           ;set STB=0                       or wraps from 2 to 0
;Thereafter, one should read the Sensitivity bits, typically like so:
[4016h]=01h           ;set STB=1  ;\another STB on/off, for invoking reading
[4016h]=00h           ;set STB=0  ;/(not sure if this part is required)
for i=11 to 0, dummy=[4016h+n], next i              ;skip first 12 bits
for i=1 to 0, sensitivity.bit(i)=[4016h+n], next i  ;read 2 sensitivity bits
;Repeat the above procedure until the desired sensitivity value is reached.
```

Caution: According to Nintendo, the internal threshold factors aren't initialized until the change-sensitivty procedure is executed at least once (ie. after power-up, or after sensing a newly connected mouse, one MUST execute the change-sensitivity procedure, EVEN if the mouse does return the desired 2bit sensitivity code).

SNES Controllers Mouse Games

The SNES Mouse is supported by many games, whereas most of them (except Mario Paint) can be also used with normal joypads.

Games that support the SNES Mouse (the list may be incomplete)

```
Acme Animation Factory
Alice Paint Adventure
Arkanoid: Doh It Again
Bishoujo Senshi Sailor Moon S Kondowa Puzzle de Oshioikiyo! (Japan only)
Brandish 2: Expert (Japan only)
BreakThru!
Civilization
Cameltry (called On The Ball in North America and the UK)
Cannon Fodder
Dai3ji Super Robot Taisen (Japan only)
Dai4ji Super Robot Taisen (Japan only)
Doom
Dokyusei 2 (Japan only)
Dragon Knight 4 (Japan only)
Eye of the Beholder
Farland Story 2 (Japan only)
Fun and Games
Galaxy Robo (Japan only)
Hiouden: Mamono-tachi tono Chikai (Japan only)
Jurassic Park (mouse MUST be in slot 2)
King Arthur's World
Koutetsu No Kishi (Japan only)
Koutetsu No Kishi 2 (Japan only)
Koutetsu No Kishi 3 (Japan only)
Lamborghini American Challenge
Lemmings 2: The Tribes
Lord Monarch (Japan only)
The Lord of the Rings
Mario and Wario (Japan only)
Mario Paint (1992)
Mario's Super Picross (Japan only)
Mario's Early Years: Pre-School
Mega Lo Mania
Might and Magic III
Motoko-chan no Wonder Kitchen (Japan only)
Nobunaga's Ambition
On the ball
Operation Thunderbolt
Pieces
Populous II
Power Monger
Revolution X
San Goku Shi Seishi: Tenbu Spirits (Japan only)
Shien's Revenge
SimAnt
Snoopy Concert
Sound Fantasy (unreleased)
Spellcraft (unreleased)
Super Caesars Palace
Super Game Boy
Super Castles (Japan only)
Super Noah's Ark 3D
Super Pachi-slot Mahjong
Super Robot Wars 3
Super Solitaire
Terminator 2: The Arcade Game
Tin Star
Tokimeki Memorial (Japan only)
Troddlers
Utopia
Vegas Stakes
Warrior of Rome III (unreleased)
Wolfenstein 3D
Wonder Project J
Zan 2: Spirits (Japan only)
Zan 3: Spirits (Japan only)
```

Plus (!):

```
Kaite Tsukutte Asoberu Dezaemon (Japan only)
Pro Action Replay Mk3
Kakinoki Shogi (1995) ASCII Corporation (JP)
Spell Craft: Aspects of Valor (1993) ... same as "Spellcraft (unreleased)"?
  (in Spellcraft: mouse works in-game only, not in title screen)
```

Moreover, the "SNES Test Program" (1991 by Nintendo) includes a Mouse test.

SNES Controllers Multiplayer 5 (MP5) (Five Player Adaptor)

The MP5 plugs into one Controller Port on the SNES (typically Port 2), and has 4 ports for controllers to be plugged into it (labeled 2 through 5). It also has an override switch which makes it pass through Pad 2 and ignore everything else.

Reading Controller Data

```
[4016h].Bit0=1                              ;-strobe on (to player 1-5)
[4016h].Bit0=0                              ;-strobe off (to player 1-5)
read any number of bits from [4016h].Bit0   ;-read Player 1 data
read any number of bits from [4017h].Bit0/Bit1 ;-read Player 2/3 data
[4201h].Bit7=0                              ;-select Player 4/5
read any number of bits from [4017h].Bit0/Bit1 ;-read Player 4/5 data
[4201h].Bit7=1  ;(prepare here for next frame) ;-select Player 2/3
do no further access until next frame (allow [4201h].Bit7=1 to stabilize)
```

The strobe on/off part, and reading first 16bits for player 1-3 is usually done via automatic reading (whereas, Player 3 data will obviously show up in "JOY4" register, not in "JOY3" register). Whilst reading further player 1-3 bits, and all player 4-5 bits is done via manual reading.
As shown above, player 2-3 should be always read before 4-5, for two reasons:

At least some MP5 devices may respond slowly on 0-to-1 transitions of [4201h].Bit7 (unless the device contains a pull-up resistor). Some MP5's (namely the Tribal Tap) are always passing CLK to player 2 (in that case player 2 data would be shifted-out when accessing player 4-5 data).

Detecting the MP5 Hardware

Below can be used to detect MP5 in ports 1 (n=0) and 2 (n=1). Games do usually check both ports (and show an error messages when sensing a MP5 in port 1).

```
[4016h].Bit0=1                          ;-strobe on (force MP5 Bit1=1)
read 8 bits from [4016h+n].Bit1 to byte A  ;-read byte A
[4016h].Bit0=0                          ;-strobe off (normal data mode)
read 8 bits from [4016h+n].Bit1 to byte B  ;-read byte B
if A=FFh and B<>FFh then MP5=present    ;-verify result
```

If there's no MP5 connected, then A and B will be typically 00h (since most controllers don't use [4017h].Bit1, exceptions are Turbo File, SFC Modem, Voice-Kun, and X-Band Keyboard).
If a MP5 is connected, then A will be FFh, and B will be first 8bit of data from joypad 3 or 5 (which can't be FFh since one can't push all four DPAD directions at once).

Also note that there is nothing preventing the MP5 from functioning perfectly when plugged in to Port 1, except that the game must use bit 6 of $4201 instead of bit 7 to set IOBit and must use the Port 1 registers instead of the Port 2 registers. With 2 MP5 units, one could actually create an 8-player game.

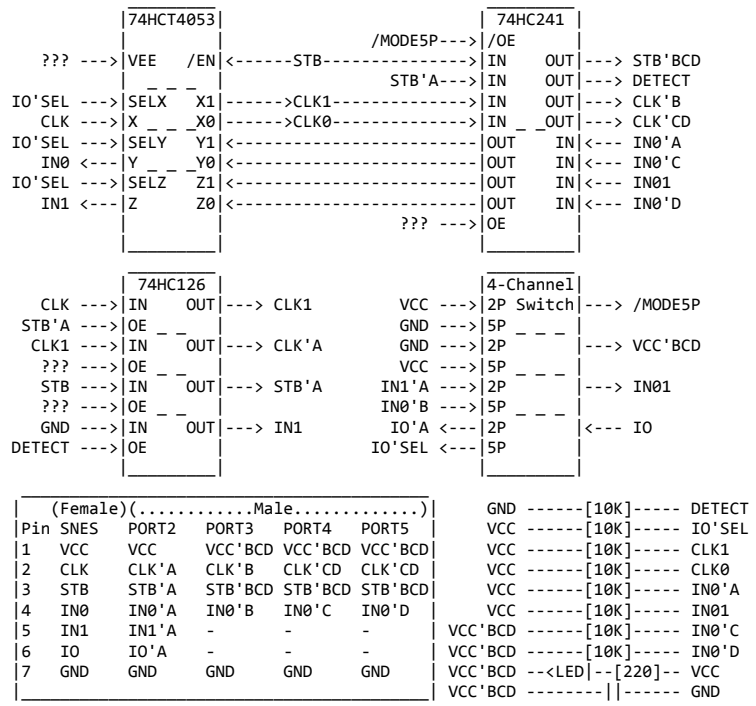
Supported/Unsupported Games/Hardware

The Multiplayer is supported by more than 100 games, but incompatible with almost everything except normal joypads.
SNES Controllers Multiplayer 5 - Unsupported Hardware
SNES Controllers Multiplayer 5 - Supported Games

Multitap/Multiplayer Adaptor Versions

```
2or3? Way Multiplay Adaptor (Gamester LMP) (with only 2 (or 3?) sockets)
5 Player Game Plug (Laing) (same polygonal case as SN-5)
HORI Multitap HSM-07 (HORI) (4 "top-loading" connectors)
HORI Super Tetris 3 (HORI) (red case, otherwise same as HORI HSM-07)
Multi Adaptor Auto (Partyroom21)
Multi Player Adaptor (unknown manufacturer) (roughly PS1 shaped)
Multi-Player Adaptor (Super Power) (same case as Multiplay Adaptor from LMP)
Multiplay Adaptor (Gamester LMP) (square gray case, "crown" shaped LMP logo)
SN-5 Multitap (Phase 9) (same polygonal case as Super 5 QJ/Super 5-Play)
SNES MultiPlayer 5 Schematic Diagram (1st May 1992) (Nintendo) (book2.pdf)
Super 5 QJ (same polygonal case as SN-5)
Super 5 Multi-Player Adapter by Innovation (same polygonal case as SN-5)
Super 5-Play (Performance) (same polygonal case as SN-5)
Super Link by BPS (Bullet Proof Software) (same case as HORI HSM-07)
Super Multitap (noname) (polyshaped, but different than the SN-5 case)
Super Multitap (Hudson) (long slim device with 4 connectors on front panel)
Super Multitap 2 (Hudson) (square device with yellow Bomberman face)
Super Multitap Honest (same polygonal case as SN-5)
Tribal-Tap 5 (Nakitek) (same case as Multiplay Adaptor from LMP)
Tribal Tap, 6 Player Adaptor (Naki)
Tribal Tap, 6 Player Adaptor (Fire) (same as Naki, but without Naki logo)
```

SNES MultiPlayer 5 - Schematic Diagram (Rev 2.3) 1st May 1992



The schematic was released by Nintendo (included in book2.pdf), components are:

```
74HCT4053 (triple 2-to-1 line analog multiplexer/demultiplexer)
74HC126 (quad 3-state noninverting buffer with active high enables)
74HC241 (dual 4-bit 3-state noninverting buffer/line driver)
4-channel 2-position switch (2P/5P-mode selection)
LED (glows in 2P-mode)
1 female joypad connector, 4 male joypad connectors
plus some resistors
```

Connection of the four "???" pins is unclear (maybe just wired to VCC or GND).
Unknown if any of the existing adaptors do actually use the above schematic.

Tribal Tap (Naki)

This adaptor is supposed to support up to 6 players (one more than the normal multitaps). The 6-player feature isn't supported by any games, and it's unknown how to access the 6th port by software - some people do believe that it isn't possible at all, and the the 6th port is just a fake - but, that theory is based on the

(incorrect) assumption that PALs cannot be programmed to act as flipflops. However, if the schematic shown below is correct (the "IN0'E" signal from Port6 being really & solely <input> to the OUT0 <output> pin; not verified), then it's probably really a fake.
The Tribal Tap schematic should be reportedly looking somehow like so:

```
                        .-----. .-----.
 VCC--[RP]-------CLK---> 1|IN0  '-'  VCC|20 <---VCC
 VCC--[RP]--------IO---> 2|IN1 16L8 OUT7|19 --->IN0    (out-only)
 VCC--[RP]-----IN0'A---> 3|IN2 PAL  I/O6|18 --->IN1
 VCC--[RP]-----IN1'A---> 4|IN3      I/O5|17 --->STB'A
 VCC--[RP]---/MODE6P---> 5|IN4      I/O4|16 --->IO'A
 VCC--[RP]-----IN0'B---> 6|IN5      I/O3|15 --->CLK'B
 VCC--[RP]---/MODE2P---> 7|IN6      I/O2|14 --->CLK'CDE
 VCC--[RP]-----IN0'D---> 8|IN7      I/O1|13 --->STB'BCDE
 VCC--[RP]-----IN0'C---> 9|IN8      OUT0|12 <---IN0'E  (out-only???)
               GND -->10|GND       IN9|11 <---/STB----[R]---VCC
                        '-------------'

     .----------.           .-------.             .-------.
     |3-position|           | S9013 |             | S9013 |        .-[R]-STB
     |switch  2P|--/MODE2P  |      E|-->VCC'BCDE  |      E|---GND  |
 GND-|        5P|--NC       | NPN  B|<--/MODE2P   | NPN  B|<-------+
     |        6P|--/MODE6P  |      C|---SNES.VCC  |      C|-->/STB |
     '----------'           '-------'             '-------'        '-[R]-GND
 .-------------------------------------------------------.
 |   (Female)(.............Male.....................)|    ???--|LED>--???
 |Pin SNES   PORT2   PORT3    PORT4    PORT5    PORT6   |
 |1   VCC    VCC     VCC'BCDE VCC'BCDE VCC'BCDE VCC'BCDE|    further resistors
 |2   CLK    CLK     CLK'B    CLK'CDE  CLK'CDE  CLK'CDE |    ???
 |3   STB    STB'A   STB'BCDE STB'BCDE STB'BCDE STB'BCDE|
 |4   IN0    IN0'A   IN0'B    IN0'C    IN0'D    IN0'E   |    (not installed)
 |5   IN1    (IN1'A) -        -        -        -       |    diodes ???
 |6   IO     (IO'A)  -        -        -        -       |
 |7   GND    GND     GND      GND      GND      GND     |
 '-------------------------------------------------------'
    Note: The PCB has wires to all 7 pins of PORT2,
    but the installed connector has only 5 pins, so,
    in practice, IN1'A and IO'A are not connected.
```

SNES Controllers Multiplayer 5 - Unsupported Hardware

The Multiplayer 5 is incompatible with almost everything except normal joypads/joysticks. The only other things that do work are Twin Taps, and maybe also the NTT Data Pad (unless it exceeds 17mA, and unless games do refuse it as device with "unknown" controller ID).

Unsupported Hardware in MP5 controller slots (due to missing signals)

```
Lightguns
Turbo File
SFC Modem
Voice-Kun
X-Band Keyboard
A second MP5 plugged into the first MP5
```

Prohibited Hardware
The hardware/combinations listed below are "prohibited" (and aren't supported by any offical/licensed games). Nethertheless, they should be working in practice, hopefully without immediately catching fire or blowing the fuse (but might act unstable or cause some overheating in some situations; results <might> vary depending on hardware/production variants, room temperature, used CPU/PPU/APU load, individual or official safety measures, and on type/amount of connected cartridges/controllers).

Prohibited Hardware in MP5 controller slots (due to exceeding 17mA per slot)

```
Mouse (requires 50mA)
Devices with unknown controller IDs (which might exceed 17mA)
Maybe also various unlicensed wireless/autofire joypads/joysticks
```

Prohibited Hardware in CARTRIDGE slot (due to overall power consumption)

```
Cartridges with GSU-n (programmable RISC CPU) (aka Super FX/Mario Chip)
Maybe also things like X-Band modem and Cheat Devices
```

Prohibited Hardware in BOTH controller ports (unspecified reason)

```
Two MP5's (connected to port 1 and 2) (maybe also power consumption related)
```

Prohibited Hardware in FIRST controller port (just by convention)

```
MP5 in port 1 (instead of port 2) (would mess-up the port 2-5 numbering)
```

SNES Controllers Multiplayer 5 - Supported Games

Multiplayer 5 Games

Game	Languages	Players
Bakukyuu Renpatsu!! Super B-Daman ("battle mode")	(J)	4
Bakutou Dochers: Bumps-jima wa Oosawagi ("battle mode")	(J)	4
Barkley Shut Up and Jam! / Barkey no Power Dunk	(E,J)	4
Battle Cross (supports only 5 joypads; player 6 always inactive)	(J)	5
Battle Jockey	(J)	4
Bill Walsh College Football	(E)	4
Chibi Maruko-chan: Mezase! Minami no Island!!	(J)	4
College Slam	(E)	4
Crystal Beans From Dungeon Explorer	(J)	3
Dino Dini's Soccer!	(E,F,G)	??
Dragon: The Bruce Lee Story	(E)	3
Dream Basketball: Dunk & Hoop (only 5, not 6)	(J)	5
Dynamic Stadium	(J)	4
Elite Soccer / World Cup Striker	(E,F,G,J)	4
ESPN National Hockey Night	(E)	5
FIFA International Soccer	(E,J)	4
FIFA Soccer 96	(E,F,G,I,S,SW)	4
FIFA Soccer 97: Gold Edition / FIFA 97: Gold Edition	(E,F,G,I,S,SW)	5
FIFA 98: Road to World Cup	(E,F,G,I,S,SW)	5

Finalset	(J)	4
Fire Striker / Holy Striker	(E,J)	4
Fever Pitch Soccer / Head-On Soccer	(E,F,G,I,S)	4
From TV Animation Slam Dunk: SD Heat Up!!	(J)	5
Go! Go! Dodge League	(J)	4
HammerLock Wrestling/Tenryuu Genichirou no Pro Wrestling Revol.	(E,J)	4
Hanna Barbera's Turbo Toons	(E)	5
Hat Trick Hero 2	(J)	4
Hebereke no Oishii Puzzle wa Irimasenka	(J)	5
Human Grand Prix III: F1 Triple Battle	(J)	3
Human Grand Prix IV: F1 Dream Battle	(J)	3
Hungry Dinosaurs / Harapeko Bakka	(E,J)	??
International Superstar Soccer Deluxe/Jikkyou World Soccer 2	(E,J)	4
J.League Excite Stage '94 / Capcom's Soccer Shootout	(E,J)	4
J.League Excite Stage '95	(J)	4
J.League Excite Stage '96	(J)	4
J.League Soccer Prime Goal	(J)	?
J.League Super Soccer '95 Jikkyo Stadium	(J)	4
J.R.R. Tolkien's The Lord of the Rings: Volume 1	(E,G)	4
Jikkyou Power Pro Wrestling '96: Max Voltage (only 4, not 5) 5?	(J)	4
Jimmy Connors Pro Tennis Tour (japanese version only?)	(E,F,G,J)	4
JWP Joshi Pro Wrestling: Pure Wrestle Queens	(J)	4
Kingyo Chuuihou! Tobidase! Game Gakuen	(J)	3
Kunio-kun no Dodge Ball Da yo Zenin Shuugou!	(J)	4
Looney Tunes B-Ball / Looney Tunes Basketball	(E)	4
Madden NFL '94 / NFL Pro Football '94	(E,J)	4
Madden NFL 95	(E)	4
Madden NFL 96	(E)	5
Madden NFL 97	(E)	4
Madden NFL 98	(E)	5
Micro Machines	(E)	4
Micro Machines 2: Turbo Tournament	(E)	4
Mizuki Shigeru no Youkai Hyakkiyakou	(J)	4
Multi Play Volleyball	(J)	4
Natsume Championship Wrestling	(E)	4
N-Warp Daisakusen (homebrew) (requires 2 multitaps)	(E)	8
NBA Give 'n Go / NBA Jikkyou Basket: Winning Dunk	(E,J)	4
NBA Hang Time	(E)	4
NBA Jam	(E,J)	4
NBA Jam: Tournament Edition	(E,J)	4
NBA Live 95	(E,J)	5
NBA Live 96	(E)	5
NBA Live 97	(E)	5
NBA Live 98	(E)	5
NCAA Final Four Basketball	(E)	4
NCAA Football	(E)	4
NFL Quarterback Club / NFL Quarterback Club '95	(E,J)	5
NFL Quarterback Club 96	(E,J)	5
NHL '94 / NHL Pro Hockey '94	(E,J)	5
NHL '94 / NHL Pro Hockey '94	(E,J)	5
NHL '95	(E)	?
NHL '96	(E)	?
NHL '97	(E)	?
NHL '98	(E)	5
Olympic Summer Games	(E)	5
Peace Keepers, The / Rushing Beat Shura	(E,J)	4
Pieces / Jigsaw Party	(E,J)	5
Rap Jam: Volume One	(E,F,S)	4
Saturday Night Slam Masters / Muscle Bomber: Body Explosion	(E,J)	4
Secret of Mana / Seiken Densetsu 2	(E,F,G,J)	3
Secret of Mana 2 / Seiken Densetsu 3 (with patch by Parlance)	(E,J,patch)	3
Shijou Saikyou no Quiz Ou Ketteisen Super (uses 4 twin taps)	(J)	8
Shin Nihon Pro Wrestling: Chou Senshi in Tokyo Dome	(J)	4
Shin Nihon Pro Wrestling Kounin: '94 Battlefield in Tokyo Dome	(J)	4
Shin Nihon Pro Wrestling Kounin: '95 Tokyo Dome Battle 7	(J)	4
Smash Tennis / Super Family Tennis	(E,J)	4
Sporting News Power Baseball, The	(E)	4
Sterling Sharpe: End 2 End	(E)	4
Street Hockey '95	(E)	4
Street Racer	(E,J)	4
Sugoi Hebereke	(J)	4
Sugoro Quest++: Dicenics	(J)	4
Super Bomberman	(E,J)	4
Super Bomberman: Panic Bomber World	(J)	4
Super Bomberman 2	(E,J)	4
Super Bomberman 3	(E,J)	5
Super Bomberman 4	(J)	5
Super Bomberman 5	(J)	5
Super Fire Pro Wrestling: Queen's Special	(J)	?
Super Fire Pro Wrestling Special	(J)	?
Super Fire Pro Wrestling X	(J)	?
Super Formation Soccer 94: World Cup Edition	(J)	?
Super Formation Soccer 95: della Serie A	(J)	?
Super Formation Soccer 96: World Club Edition	(J)	?
Super Formation Soccer II	(J)	?
Super Ice Hockey / Super Hockey '94	(E,J)	?
Super Kyousouba: Kaze no Sylphid	(J)	?
Super Power League	(J)	?
Super Puyo Puyo Tsu: Remix	(J)	4
Super Slam Dunk / Magic Johnson no Super Slam Dunk!	(E,J)	?
Super Tekkyuu Fight!	(J)	?
Super Tetris 3	(J)	4
Syndicate	(E,F,G,J)	4
Tiny Toon Adventures: Wacky Sports/Dotabata Daiundoukai	(E,J)	4
Top Gear 3000 / Planet's Champ TG 3000, The	(E,J)	4
Vegas Stakes / Las Vegas Dream in Golden Paradise	(E,J)	4
Virtual Soccer / J.League Super Soccer	(E,J)	5
Vs. Collection	(J)	?
Wedding Peach	(J)	3
WWF Raw	(E)	4
Yuujin no Furi Furi Girls	(J)	4
Zero 4 Champ RR	(J)	?

```
 Zero 4 Champ RR-Z                                          (J)        ?
```

Plus...

```
 Momotaro Dentetsu Happy (1996) Hudson Soft (JP)
 Bomberman B-Daman: 4 players (via Battle)
 Kiteretsu Daihyakka - Choujikuu Sugoroku: 5 players
 J.League '96 Dream Stadium: 4 players
```

Corrections:

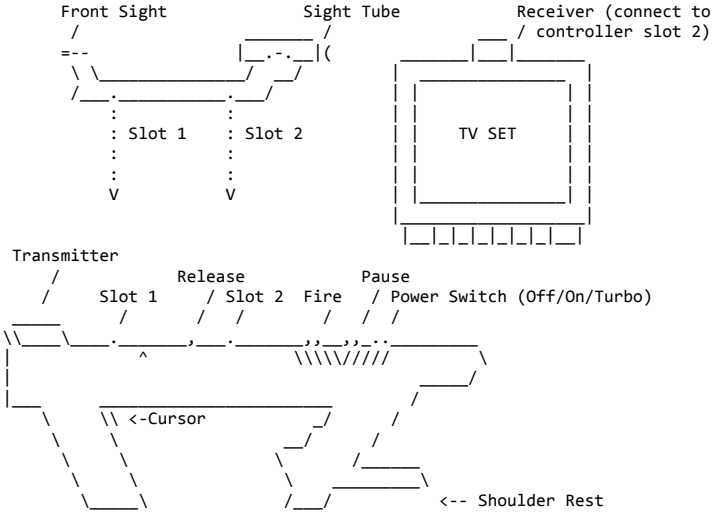
```
 Elite Soccer/World Cup Striker: 5 players (Using the "Multiple Players"
   option while setting up for a game, you can assign different people
   to different controllers.)
 FIFA International Soccer: 5 players
 FIFA Soccer 96: This actually supports 5 players, not four. Pause the game
   and access the controllers with a multi-tap connected... up to five can
   join in.
 Madden NFL '94: 5 players
 Madden NFL '95: 5 players
 Madden NFL '97: 5 players
```

Does this really support the multi-tap?

```
  Dino Dini's Soccer
  J.League Soccer Prime Goal
  NHL '95 through '97
```

SNES Controllers SuperScope (Lightgun)

SUPER SCOPE



Batteries

Takes six "AA" batteries. Which do reportedly last only for a few hours.

Super Scope Bits

```
  1st          Fire Button   (0=High=Released, 1=Low=Pressed/Newly Pressed)
  2nd          Cursor Button (0=High=Released, 1=Low=Pressed)
  3rd          Turbo Switch  (0=Normal/PowerOn, 1=Turbo/PowerOn)
  4th          Pause Button  (0=High=Released, 1=Low=Newly Pressed)
  5th..6th     Extra ID Bits (always 0=High)
  7th          Offscreen     (0=High=Okay, 1=Low=CRT-Transmission Error)
  8th          Noise         (0=High=Okay, 1=Low=IR-Transmission Error)
  9th..12th    Extra ID Bits (always 1=Low)
  13th..16th   ID Bits       (always 1=Low=One) (0Fh)
  17th and up  Unused        (always 1=Low)
```

'f' is Fire, 'c' is Cursor, 't' is Turbo, 'p' is Pause, 'o' is Offscreen, and 'n' is Noise.

For obtaining the H/V position& latch flag (Ports 213Ch,213Dh,213Fh), see:
SNES PPU Timers and Status

The SuperScope has two modes of operation: normal mode and turbo mode. The current mode is controlled by a switch on the unit, and is indicated by the 't' bit. Note however that the 't' bit is only updated when the Fire button is pressed (i.e. the 'f' bit is set). Thus, when you turn turbo on the 't' bit remains clear until you shoot, and similarly when turbo is deactivated the bit remains set until you fire.

In either mode, the Pause bit will be set for the first strobe after the pause button is pressed, and then will be clear for subsequent strobes until the button is pressed again. However, the pause button is ignored if either cursor or fire are down(?).

In either mode, the Cursor bit will be set while the Cursor button is pressed.

In normal mode, the Fire bit operates like Pause: it is on for only one strobe. In turbo mode, it remains set as long as the button is held down.

When Fire/Cursor are set, Offscreen will be set if the gun did not latch during the previous strobe and cleared otherwise (Offscreen is not altered when Fire/Cursor are both clear).

If the Fire button is being held when turbo mode is activated, the gun sets the Fire bit and begins latching. If the Fire button is being held when turbo mode is deactivated, the next poll will have Fire clear but the Turbo bit will not be updated until the next fire (i.e. FcTp => turbo off => fcTp, not fctp).

The PPU latch operates as follows: When Fire or Cursor is set, IOBit is set to 0 when the gun sees the TV's electron gun, and left a 1 otherwise. Thus, if the SNES also leaves it one (bit 7 of $4201), the PPU Counters will be latched at that point. This would also imply that bit 7 of $4213 will be 0 at the moment the SuperScope sees the electron gun.

Since the gun depends on the latching behavior of IOBit, it will only function properly when plugged into Port 2. If plugged into Port 1 instead, everything will work except that there will be no way to tell where on the screen the gun is pointing.

When creating graphics for the SuperScope, note that the color red is not detected. For best results, use colors with the blue component over 75% and/or the green component over 50%.

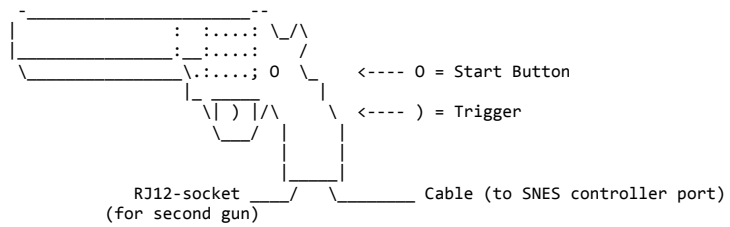
Data2 is presumably not connected, but this is not known for sure.

Games compatible with the Super Scope

```
Battle Clash (US) (EU) / Space Bazooka (JP) (1992) Nintendo
Bazooka Blitzkrieg (US) (1992) Bandai
Hunt for Red October (used for bonus games) (US) (EU) (JP)
Lamborghini American Challenge (used in special game mode) (US) (EU) (1993)
Lemmings 2 (US) (EU) (JP) (1994) Psygnosis (at game start: aim at crosshair)
Metal Combat: Falcon's Revenge (includes OBC1 chip) (US) (1993)
Operation Thunderbolt (US) (1994) Taito
Super Scope 6 (bundled with the hardware) (Blastris & LazerBlazer) (1992)
Terminator 2 - T2: The Arcade Game (US) (EU) (JP) (1993) Carolco/LJN
Tin Star (US) (1994) Nintendo
X-Zone (US) (EU) (1992) Kemco
Yoshi's Safari (US) (EU) (JP) (1993) Nintendo
```

Moreover, the "SNES Test Program" (1991 by Nintendo) includes a Super Scope test, and, reportedly, there's also a special Super Scope test cartridge (?)

SNES Controllers Konami Justifier (Lightgun)




```
Blue Gun --> connects to SNES (and has 6pin RJ12 socket for second gun)
Pink Gun --> connects to 6pin RJ12 socket of first gun
```

Justifier Bits

```
1st..12th   Unused             (always 0=High)
13th..16th  ID Bit3-0          (MSB first, 1=Low=One, always 0Eh = 1110b)
17th..24th  Extra ID Bit7-0    (MSB first, 1=Low=One, always 55h = 01010101b)
25th        Gun 1 Trigger      (1=Low=Pressed?)
26th        Gun 2 Trigger      (1=Low=Pressed?)
27th        Gun 1 Start Button (1=Low=Pressed?)
28th        Gun 2 Start Button (1=Low=Pressed?)
29th        Previous Frame was H/V-latching Gun1/2 (1=Low=Gun1, 0=High=Gun2)
30th..32th  Unused             (always 0=High)
33th and up Unused             (always 1=Low)
```

SNES Justifier Game(s)

```
Lethal Enforcers (bundled with the hardware) (1993) Konami (US) (EU) (JP)
```

Sega Version

There's also an identically looking Blue Gun for Sega (but with 9pin joystick connector). The Pink Gun can be used with both SNES and Sega Blue Gun versions.

The Justifier returns 48 bits of data out Data1. Presumably it returns one bits after (if so, it really only returns 32 bits), but this is not known. The data is:

```
0000000000001110 01010101TtSsl000 1111111111111111
```

T/t are the trigger states for guns 1 and 2. S/s are the start button states for guns 1 and 2. 'l' indicates which gun was connected to IOBit: 1 means gun 1, 0 means gun 2. Note that 'l' toggles even when gun 2 is not connected.

IOBit is used just like for the SuperScope. However, since two guns may be plugged into one port, which gun is actually connected to IOBit changes each time Latch cycles. Also note, the Justifier does not wait for the trigger to be pulled before attempting to latch, it will latch every time it sees the electron gun. Bit 6 of $213F may be used to determine if the Justifier was pointed at the screen or not.

Data2 is presumably not connected, but this is not known for sure.

For obtaining the H/V position& latch flag (Ports 213Ch,213Dh,213Fh), see:
SNES PPU Timers and Status

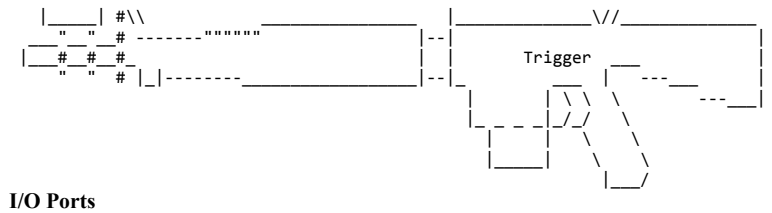
Nardz: "Actually when I was a kid, I bought the SNES Justifier Battle Clash package. The Weird thing about it is that The package had A Sega Justifier in it, but it came with this SNES/Sega Adapter, which pluged into your SNES and the Sega Justifier would plug into the back of the connector." -- But, Battle Clash is a Super Scope game, not a Justifier game???

The pinouts of the 6pin RJ12-socket are unknown.

SNES Controllers M.A.C.S. (Lightgun)

Multi-Purpose Arcade Combat Simulator (M.A.C.S.)

This lightgun was used by the US Army to introduce beginners how to kill real people. It was also shown on career days at high schools to win new recruits. The hardware consists of a small lightpen attached to a M16 rifle. Software cartridges exist for C64 and SNES.



I/O Ports
The lightgun connects to the lightpen input on 2nd controller port. Aside from the HV-Latches, it uses only one I/O signal:
```
4017h.Bit0 Trigger Button (1=LOW=Pressed)
```
That is, only a single bit (no serial data transfer with CLK/STB signals). A standard joypad attached to 1st controller port allows to calibrate the lightpen (via Select button).
For obtaining the H/V position& latch flag (Ports 213Ch,213Dh,213Fh), see:
SNES PPU Timers and Status

SNES Software
```
MACS Basic Rifle Marksmanship v1.1e (v1.2a) (1993) Sculptured Software (US)
MACS Basic Rifle Marksmanship v1994.0 (1994?)
MACS Moving Target Simulator (?)
```
Note: Version "1.1e" is displayed in the title screen, whilst the version string at ROM offset 05819h identifies it as version "1.2a". The program code looks crude and amateurish, and (as indicated by the corrupted ROM header) it never passed through Nintendo's Seal of Quality process.

SNES Controllers Twin Tap

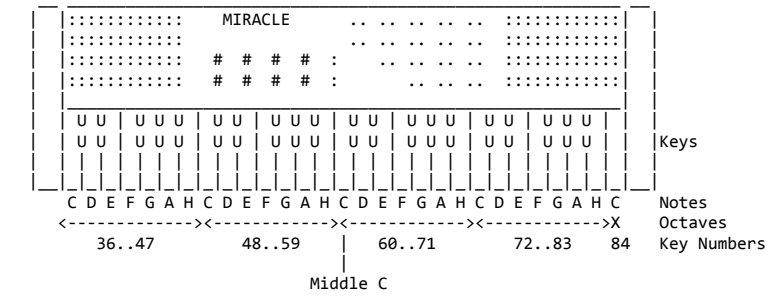
The Twin Tap from Partyroom21 (aka Yonezawa PR21) is a special controller for 8-player quiz games. The Twin Tap itself consists of 7pin SNES controller connector with two cables, and a push-button on each cable-end (one button per player). For the 8-player mode, four Twin Taps need to be connected to a multiplayer adaptor (such like Partyroom21's own "Multi Adaptor Auto").

The Twin Tap is supported by
```
Shijou Saikyou no Quiz Ou Ketteisen Super (1992) TBS/Partyroom21/S'pal (JP)
```

Transfer Protocol
```
1st         Button 2 (or 4/6/8) (1=Low=Pressed) (would be "B" on joypads)
2nd         Button 1 (or 3/5/7) (1=Low=Pressed) (would be "Y" on joypads)
3rd..12th   Unknown
13th..16th  Unknown (would be ID Bit3..0 on other SNES controllers)
17th..24th  Unknown (would be Extended ID Bit7..0 on other SNES controllers)
25th and up Unknown
```
Judging from disassembled game code, the 4bit ID might be 00h or 0Eh, in the latter case, there <should> be also a unique Extended ID value.

SNES Controllers Miracle Piano

Miracle Piano Teaching System (The Software Toolworks)
SNES Controllers Miracle Piano Controller Port
SNES Controllers Miracle Piano MIDI Commands
SNES Controllers Miracle Piano Instruments
SNES Controllers Miracle Pinouts and Component List




```
49 Piano Keys (29 White Keys, 20 Black Keys) (4 octaves, plus next higher C)
1  Foot Pedal (Sustain)
8  Push Buttons (Mode/Volume Selection)
```

SNES Controllers Miracle Piano Controller Port

Miracle Controller Port Transfer
Read Direction (invoked by SHORT Strobe signal)
```
1st         Data Present Flag (0=High=None, 1=Low=Yes)
2nd..9th    Data Bit7..0      (MSB First, inverted 1=LOW=Zero)
10th..12th  Unknown
13th..16th  Unknown (would be ID Bit3..0 on other SNES controllers)
17th and up Unknown
```
Write Direction (invoked by LONG Strobe signal, data output on STROBE line)
```
1st..8th    Data Bit7..0      (MSB First, 0=LOW=Zero)
```
Observe that read/write direction depends on length of initial Strobe signal (so games that are reading joypad with other strobe-lengths might mess up things). 10th bit and up (including the 4bit Controller ID) might be garbage (depending on how the 8051 CPU in the keyboard handles the data transfer). However, with appropriate timings, detecting a Miracle could be done via the "Firmware version request" MIDI command.
Note: The NES and SNES Miracle software expects the piano keyboard connected to Port 1, and a normal joypad connected to Port 2.

miracle_recv_byte
```
[004016h]=01h                           ;strobe on
delay (strobe=1 for 102 master clks)    ;short delay = READ mode
[004016h]=00h                           ;strobe off
data_present_flag = [004016h].bit0      ;data present flag (1=LOW=Yes)
```

```
  for i=7 to 0
    data.bit(i)=NOT [004016h].bit0          ;data bits (MSB first, 1=LOW=Zero)
  next i
```

miracle_send_byte

```
  [004016h]=01h                             ;strobe on (start bit)
  delay (strobe=1 for 528 master clks)      ;long delay = WRITE mode
  for i=7 to 0
    [004016h].bit0=data.bit(i)              ;data bits (MSB first, 1=HIGH=One)
    dummy=[004016h]                         ;issue short CLK pulse
  next i
  [004016h]=00h                             ;strobe off (stop/idle)
  delay (strobe=0 for min 160 master clks)  ;medium delay
```

SNES Controllers Miracle Piano MIDI Commands

The Miracle is always using MIDI messages (no matter if the messages are transferred through MIDI or RS232 or NES/SNES/Genesis controller cables). Below lists the supported MIDI messages (including "Undocumented" messages, which are used by the Miracle's SNES software, although they aren't mentioned in the Miracle's Owner's Manual).

MIDI Information Sent FROM/TO The Miracle keyboard

```
  Expl.                     Dir  Hex
  Note off (Undocumented)     W  8#h,<key>,00h   ;same as Note ON with velo=00h
  Note on/off command       R/W  9#h,<key>,<velo>
  Main volume level           W  B0h,07h,<vol>
  Sustain on/off command    R/W  B#h,40h,<flag>
  Local control on/off        W  B0h,7Ah,<flag>
  All notes off               W  B#h,7Bh,00h
  Patch change command (*)  R ?? C#h,<instr>     ;TO keyboard = Undocumented
  Miracle button action     R    F0h,00h,00h,42h,01h,01h,<bb>,F7h
  Unknown (Undocumented)      W  F0h,00h,00h,42h,01h,02h,<??>,F7h   ;???
  Keyboard buffer overflow  R    F0h,00h,00h,42h,01h,03h,01h,F7h
  Midi buffer overflow      R    F0h,00h,00h,42h,01h,03h,02h,F7h
  Firmware version request    W  F0h,00h,00h,42h,01h,04h,F7h
  Miracle firmware version  R    F0h,00h,00h,42h,01h,05h,<maj>,<min>,F7h
  Patch split command         W  F0h,00h,00h,42h,01h,06h,0#h,<lp>,<up>,F7h
  Unknown (Undocumented)      W  F0h,00h,00h,42h,01h,07h,F7h        ;???
  All LEDs on command         W  F0h,00h,00h,42h,01h,08h,F7h
  LEDs to normal command      W  F0h,00h,00h,42h,01h,09h,F7h
  Reset (Undocumented)        W  FFh
```

Direction: R=From keyboard, W=To keyboard

Notes: (*) Patch change FROM Keyboard is sent only in Library mode.

```
  N#h         Hex-code with #=channel (#=0 from keyb, #=0..7 to keyb)
  <key>       Key (FROM Miracle: 24h..54h) (TO Miracle: 18h..54h/55h?)
  <velo>      Velocity (01h..7Fh, or 00h=Off)
  <vol>       Volume (00h=Lowest, 7Fh=Full)
  <flag>      Flag (00h=Off, 7Fh=On)
  <instr>     Instrument (00h..7Fh) for all notes
  <lp>        Instrument (00h..7Fh) for notes 24?/36-59, lower patch number
  <up>        Instrument (00h..7Fh) for notes 60-83/84?, upper patch number
  <maj>.<min> Version (from version 1.0 to 99.99)
  <bb>        button on/off (bit0-2:button number, bit3:1=on, bit4-7:zero)
```

Data from piano is always sent on first channel (#=0). Sending data to piano can be done on first 8 channels (#=0..7), different instruments can be assigned to each channel. Although undocumented, the SNES software does initialize 16 channels (#=0..0Fh), unknown if the hardware does support/ignore those extra channels (from the instrument table: it sounds as if one could use 16 single-voice channels or 8 dual-voice channels).

SNES Controllers Miracle Piano Instruments

Available Patches (aka Instruments)

The following patches are available through both Library Select Mode and MIDI control:

```
  000 Grand Piano     032 Marimba         064 Synth Bells    096 Tube Bells'
  001 Detuned Piano   033 Glockenspiel'   065 Vox 1          097 Frogs/Ducks
  002 FM Piano        034 Kalimba'        066 Vox 2          098 Banjo'
  003 Dyno            035 Tube Bells      067 Vox 3          099 Shakuhachi'
  004 Harpsichord     036 Steel Drums     068 Mod Synth      100 Piano'
  005 Clavinet        037 Log Drums'      069 Pluck Synth    101 Vibraphone'
  006 Organ           038 Strings 1       070 Hard Synth     102 FM Piano'
  007 Pipe Organ      039 Pizzicato       071 Syntar         103 Clock Belis'
  008 Steel Guitar    040 Strings 2       072 Effects 1 *    104 Harpsichord'
  009 12-StringGuitar 041 Violin 1'       073 Effects 2 *    105 Clavinet'
  010 Guitar          042 Trumpet'        074 Percussion 1 * 106 Organ'
  011 Banjo           043 Trumpets        075 Percussion 2 * 107 Pipe Organ'
  012 Mandolin        044 Horn'           076 Percussion 3 * 108 Metal Guitar'
  013 Koto'           045 Horns           077 Sine Organ'    109 Stick'
  014 Jazz Guitar'    046 Trombone'       078 Organ #        110 Guitar'
  015 Clean Guitar'   047 Trombones       079 Pipe Organ #   111 Xylophone'
  016 Chorus Guitar   048 CupMuteTrumpet' 080 Harpsichord #  112 Marimba'
  017 Fuzz Guitar     049 Sfz Brass 1     081 Synth Pad 1    113 Syn Trombone'
  018 Stop Guitar     050 Sfz Brass 2     082 Synth Pad 2    114 Syn Trumpet'
  019 Harp'           051 Saw Synth       083 Synth Pad 3    115 Sfz Brass 1'
  020 Detuned Harp    052 Tuba'           084 Synth Pad 4    116 Sfz Brass 2'
  021 Upright Bass'   053 Harmonica       085 Synth Pad 5    117 Saw Synth'
  022 Slap Bass'      054 Flute'          086 Synth Pad 6    118 Church Bells'
  023 Electric Bass'  055 Pan Flute'      087 Synth Pad 7    119 Marcato'
  024 Moog            056 Calliope        088 Synth Pad 8    120 Marcato
  025 Techno Bass     057 Shakuhachi      089 Synth Pad 9    121 Violin 2'
  026 Digital Waves   058 Clarinet'       090 Synth Pad 10   122 Strings 3
  027 Fretless Bass'  059 Oboe'           091 Synth Pad 11   123 Synth Bells'
  028 Stick Bass      060 Bassoon'        092 Synth Pad 12   124 Techno Bass'
  029 Vibraphone      061 Sax'            093 Synth Pad 13   125 Mod Synth'
  030 MotorVibraphone 062 Church Bells    094 Synth Pad 14   126 Pluck Synth'
  031 Xylophone       063 Big Bells       095 Synth Pad 15   127 Hard Synth'
```

Notes:

```
 ' These programs are single voice, which lets The Miracle play up to 16
   notes simultaneously. All other programs are dual voice, which lets it
```

```
  play up to 8 notes simultaneously.
* 072..076 See below for a list of Effects/Percussion sounds.
# 078..080 To be true to the nature of the sampled instrument, these patches
  do not respond to velocity.
```

Effects and Percussion Patches

When selecting instruments 072..076 (Effects 1-2 and Percussion 1-3), a number of different sounds are mapped to each six keyboard keys/notes:

Note	Effects 1	Effects 2	Percussion 1	Percussion 2	Percussion 3
30-35	Jet	Yes (ding)	-	-	Ratchet
36-41	Gunshot	No (buzz)	Kick Drum	Rim Shot	Snap 1
42-47	RoboDeath	Applause	Snare	Exotic	Snap 2
48-53	Whoosh	Dogbark	Toms	Congas	Dripdrum 1
54-59	Punch	Door creak	Cymbal	Timbale	Dripdrum 2
60-65	Slap	Door slam	Closed Hat	Cowbell	Wet clink
66-71	Duck	Boom	Open Hat	Bongos	Talk Drum
72-77	Ow! 1	Car skid	Ride	Whistle	Agogo
78-83	Ow! 2	Goose	Shaker	Clave	Explosion

Note: The piano keys are numbered 36..84 (so notes 30..35 can be used only through MIDI messages, not via keyboard).

SNES Controllers Miracle Pinouts and Component List

25pin SUBD connector (J6)

```
1  PC/Amiga/Mac RS232 GND (also wired to RTS)
2  PC/Amiga/Mac RS232 RxD
3  PC/Amiga/Mac RS232 TxD
7  NES/SNES/Genesis GND
10 NES/SNES/Genesis Data
13 NES/SNES/Genesis Strobe
14 Sense SENSE0 (0=MIDI Output off, 1=MIDI Output on)
15 Sense SENSE1 (0=9600 Baud; for RS232, 1=31250 Baud; for MIDI)
19 NES/SNES/Genesis Clock
all other pins = not connected
```

For PC/Mac RS232 wire SENSE0=GND, SENSE1=GND

Miracle NES and SNES Cartridges

According to the ROM Headers: The SNES cartridge contains 512Kbyte Slow/LoROM, and no SRAM (nor other storage memory). The NES cartridge contains MMC1 mapper, 256Kbyte PRG-ROM, 64Kbyte CHR-ROM, and no SRAM (nor other storage memory).

Miracle Piano Component List (Main=Mainboard Section, Snd=Sound Engine)

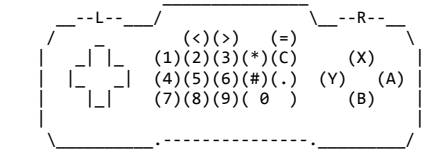
```
U1   Snd  16pin TDA7053 (stereo amplifier for internal speakers)
U2   Snd   8pin NE5532 (dual operational amplifier)
U3   Snd  16pin LM13700 or LM13600 (unclear in schematic) (dual amplifier)
U4   Snd  14pin LM324 (quad audio amplifier)
U5   Main  3pin LM78L05 (converts +10V to VLED, supply for 16 LEDs)
U6   Main 14pin 74LS164 serial-in, parallel-out (to 8 LEDs)
U7   Main 14pin 74LS164 serial-in, parallel-out (to another 8 LEDs)
U8   Main  5pin LM2931CT (converts +12V to +10V, and supply for Power LED)
U9   Main  3pin LM78L05 (converts +10V to +5REF)
U10  Snd  14pin TL084 (JFET quad operational amplifier)
U11  Snd  40pin J004 (sound chip, D/A converter with ROM address generator)
U12  Snd  32pin S631001-200 (128Kx8, Sound ROM for D/A conversion)
U13  Main  3pin LM78L05 (converts +10V to VCC, supply for CPU and logic)
U14  Main 40pin AS0012 (ASIC) Keyboard Interface Chip (with A/D for velocity)
U15  Main 40pin 8032 (8051-compatible CPU) (with Y1=12MHz)
U16  Snd  40pin AS0013 (ASIC)
U17  Main 28pin 27C256 EPROM 32Kx8 (Firmware for CPU)
U18  Main 28pin 6264 SRAM 8Kx8 (Work RAM for CPU)
U19  Main 16pin LT1081 Driver for RS232 voltages
U20  Main  8pin 6N138 opto-coupler for MIDI IN signal
S1-8 Main  2pin Push Buttons
S9   Main  3pin Power Switch (12V/AC)
J1   Main  3pin 12V AC Input (1 Ampere)
J2   Main  2pin Sustain Pedal Connector (polarity is don't care)
J3   Snd   2pin RCA Jack Right
J4   Snd   2pin RCA Jack Left
J5   Snd   5pin Headphone jack with stereo switch (mutes internal speakers)
J6   Main 25pin DB25 connector (RS232 and SNES/NES/Genesis controller port)
J7   Main  5pin MIDI Out (DIN)
J8   Main  5pin MIDI In (DIN)
JP1  Main 16pin Keyboard socket right connector
JP2  Main 16pin Keyboard socket left connector
JP3  Snd   4pin Internal stereo speakers connector
```

Note: The official original schematics are released & can be found in internet.

SNES Controllers NTT Data Pad (joypad with numeric keypad)

Special joypad with numeric keypad, for use with SFC Modem:
SNES Add-On SFC Modem (for JRA PAT)

NTT Data Controller Pad - 27-buttons (including the 4 direction keys)



NTT Data Controller Pad Bits

```
1st   Bit31  Button (B)        (0=High=Released, 1=Low=Pressed)
2nd   Bit30  Button (Y)        (0=High=Released, 1=Low=Pressed)
3rd   Bit29  Button (<) Select (0=High=Released, 1=Low=Pressed)
4th   Bit28  Button (>) Start  (0=High=Released, 1=Low=Pressed)
```

```
  5th   Bit27  Direction Up      (0=High=Released, 1=Low=Pressed)
  6th   Bit26  Direction Down    (0=High=Released, 1=Low=Pressed)
  7th   Bit25  Direction Left    (0=High=Released, 1=Low=Pressed)
  8th   Bit24  Direction Right   (0=High=Released, 1=Low=Pressed)
  9th   Bit23  Button (A)        (0=High=Released, 1=Low=Pressed)
  10th  Bit22  Button (X)        (0=High=Released, 1=Low=Pressed)
  11th  Bit21  Button (L)        (0=High=Released, 1=Low=Pressed)
  12th  Bit20  Button (R)        (0=High=Released, 1=Low=Pressed)
  13th  Bit19  ID Bit3           (always 0=High)
  14th  Bit18  ID Bit2           (always 1=Low)
  15th  Bit17  ID Bit1           (always 0=High)
  16th  Bit16  ID Bit0           (always 0=High)
  17th  Bit15  Button (0)        (0=High=Released, 1=Low=Pressed)
  18th  Bit14  Button (1)        (0=High=Released, 1=Low=Pressed)
  19th  Bit13  Button (2)        (0=High=Released, 1=Low=Pressed)
  20th  Bit12  Button (3)        (0=High=Released, 1=Low=Pressed)
  21th  Bit11  Button (4)        (0=High=Released, 1=Low=Pressed)
  22th  Bit10  Button (5)        (0=High=Released, 1=Low=Pressed)
  23th  Bit9   Button (6)        (0=High=Released, 1=Low=Pressed)
  24th  Bit8   Button (7)        (0=High=Released, 1=Low=Pressed)
  25th  Bit7   Button (8)        (0=High=Released, 1=Low=Pressed)
  26th  Bit6   Button (9)        (0=High=Released, 1=Low=Pressed)
  27th  Bit5   Button (*)        (0=High=Released, 1=Low=Pressed)
  28th  Bit4   Button (#)        (0=High=Released, 1=Low=Pressed)
  29th  Bit3   Button (.) Dot    (0=High=Released, 1=Low=Pressed)
  30th  Bit2   Button (C) Clear  (0=High=Released, 1=Low=Pressed)
  31th  Bit1   Unknown/Unused    (unknown, probably always 1 or always 0)
  32th  Bit0   Button (=) End    (0=High=Released, 1=Low=Pressed)
  33th and up  Padding           (unknown, probably always 1 or always 0)
```

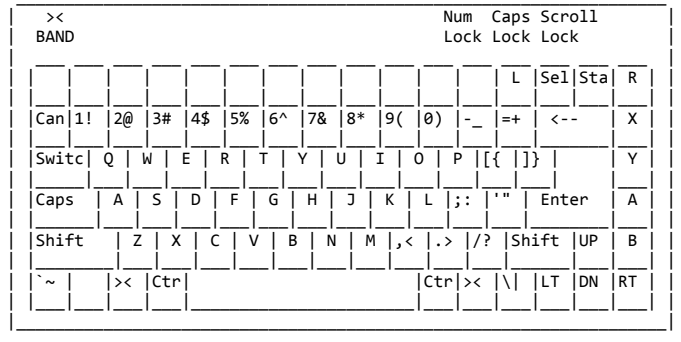
Note: The "(=)" button is sunken, somewhat preventing accidently pressing it.

SNES Controllers X-Band Keyboard

The X-Band keyboard is a (rare) optional add-on for the X-Band Modem, intended to allow faster chatting/mailing as with the joypad controlled on-screen keyboard.

Keyboard Layout

The keyboard has a black case, 84 keys, an X-Band logo in upper left, and connection cable (to SNES controller port) attached in upper-right.



Normal Controller Access (via STB and DATA0)

```
  1st-12th    Unknown/unused
  13th-16th   Unknown/unused (should be usually a 4bit ID)
  17th-24th   Unknown/unused (should be sometimes extended 8bit ID)
  25th and up Unknown/unused
```

Note: If the keyboard data is transferred in sync with STB, then "17th and up" are the LSBs of the 2bit keyboard data pairs (though it might also be in sync with falling IOBIT, rather than with STB).

Keyboard Access (read_scancodes) (via IOBIT and DATA0 & DATA1)

Below might be required to be preceeded by reading normal 16bit controller data (eg. via auto-joypad-reading) (ie. unknown if below needs leading STB signal, and preceeding 16xCLK signals).

```
  [004201]=7Fh                ;-set IOBIT=0
  id=getbits(8)               ;-read ID byte (must be 78h, aka "x")
  if CAPS=OFF [004201]=FFh    ;-set IOBIT=1 (when CAPS=OFF) (CAPS LED)
  num=getbits(4)              ;-read num scancodes
  if num>0 then for i=1 to num ;\
    [dst]=getbits(8)          ; read that scancodes (if any)
    dst=dst+1                 ;
  next i                      ;/
  [004201]=FFh  ;set IOBIT=1  ;-set IOBIT=1
```

Note: When reading the ID bits, BIOS sets IOBIT=1 after reading the first 2bit group (purpose unknown). When reading ONLY the ID (without reading following scancodes), then the scancodes do remain in the keyboard queue.

getbits(n)

```
  for i=1 to n/2                    ;\
    delay (loop 7 times or so)      ; read 2bits at once, LSB first
    x=(x SHR 2) OR ([004017h] SHL 6)  ;
  next                              ;/
  x=(x XOR FFh) SHR (8-n)           ;-invert & move result to LSBs
```

Scancode Summary

```
  nn              normal key
  F0h,nn          normal key released
  E0h,nn          special key
  E0h,F0h,nn      special key released
```

Normal Scancodes (without prefix)

```
  ____0xh____1xh___2xh___3xh___4xh___5xh___6xh___7xh_____8xh_____9xh___
  x0h ---    ---   ---   ---   ---   ---   ---   NUM-0   ---     <90h>?
  x1h ---    CTR1? C     N     .<    ---   ---   NUM-.   ---     <91h>?
```

```
x2h ---    SHF1? X    B    K    '"   ---  NUM-2   ---     <92h>?
x3h ---    ---   D    H    I    ---  ---  NUM-5   ---     <93h>?
x4h ---    CTR2? E    G    O    [{   ---  NUM-6   NUM-SUB <94h>?
x5h ---    Q     4$   Y    0)   =+   ---  NUM-8   ---     <95h>?
x6h ---    1!    3#   6^   9(   ---  BS   CANCEL  JOY-A   <96h>?
x7h ---    ---   ---  ---  ---  ---  ---  NUM-DIV JOY-B   <97h>?
x8h ---    ---   ---  ---  ---  CAPS ---  ---     JOY-X   <98h>?
x9h ---    ---   SPACE ---  .>   SHF2? NUM-1 NUM-RET JOY-Y   <99h>?
xAh ---    Z     V    M    /?   ENTER ---  NUM-3   JOY-L   <9Ah>?
xBh ---    S     F    J    L    ]}   NUM-4 ---     JOY-R   <9Bh>?
xCh ---    A     T    U    ;:   \|   NUM-7 NUM-ADD SELECT  <9Ch>?
xDh SWITCH W     R    7&   P    \|   ---  NUM-9   START   <9Dh>?
xEh `~     2@    5%   8*   -_   ---  ---  NUM-MUL <8Eh>?  ---
xFh ---    ---   ---  ---  ---  ---  ---  ---     <8Fh>?  ---
```

Special Scancodes (with E0h-prefix)

```
E0h,5Ah  JOY-A (alternate to normal scancode 86h)
E0h,6Bh  LEFT
E0h,72h  DOWN
E0h,74h  RIGHT
E0h,75h  UP
```

Notes

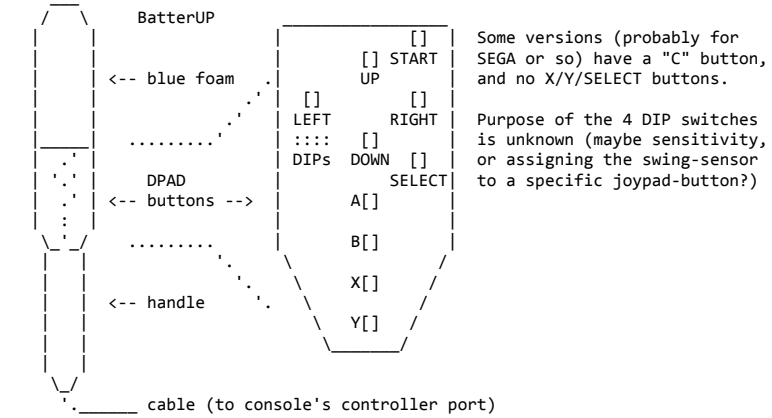
The Numeric-Keypad (NUM) isn't present on the existing X-Band keyboard. There is probably only one of the two backslash keys (5Ch/5Dh) and only one of the two Button-A keys (86h/E0h,5Ah) implemented.
There are three keyboard LEDs (Num,Caps,Scroll) visible in upper right on some photos (not visible on other photos; either due to bad photo quality, or maybe some keyboards have no LEDs). The Caps LED is controlled via software (unknown if Num/Scroll LEDs can be controlled, too).
There are several "unused" keys (which aren't used by the BIOS), unknown if/which scancodes are assigned to them (12 noname keys in upper left, 1 noname key in lower left, two control keys, and two xband logo keys). For the two shift keys it's also unknown which one has which scancode.
Unknown if the japanese BIOS includes support for japanese symbols, and unknown if there was a keyboard released in japan. (Note: With an emulated US keyboard, the Japanese BIOS does realize cursor/enter keys, it does also store typed ASCII characters in a ring-buffer at [3BB6+x]; but, for whatever reason, does then ignore those characters).

SNES Controllers Tilt/Motion Sensors

There are a few SNES controllers with Tilt/Motion Sensors, most or all of them are emulating normal SNES joypad button/direction signals, which is making them compatible with existing games, but also means that the SNES receives only digital data (pressed/released) rather than anlogue (slow/fast) data. Alltogether, the controllers appear to be more uncomfortable than useful.

BatterUP (1994) (Sports Sciences Inc.)

The BatterUP is a 24-inch foam-covered plastic baseball bat for Sega Genesis and Super Nintendo. Reportedly, it "doesn't sense swing speed or location, only timing" (whatever that means, probably simulating a button-pressed signal at the time when swinging the bat). Aside from the swing/motion sensor, the middle of the bat contains joypad buttons.

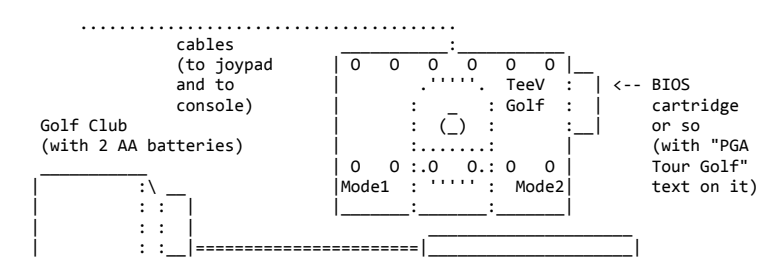


Games compatible with the SNES version (according to instruction manual?):

```
Cal Ripken Jr. Baseball, 1992 Mindscape
ESPN Baseball Tonight, 1994 Sony Imagesoft
Hardball III, 1994 Accolade
Ken Griffey Jr. Presents Major League Baseball, 1994 Nintendo
Ken Griffey, Jr.'s Winning Run, 1996 Nintendo
MLBPA Baseball, 1994 EA Sports
Sports Illustrated Championship Football and Baseball, 1993 Malibu Games
Super Baseball, 1994 EA Techmo
Super Batter Up, 1993 Namco
```

TeeV Golf (1993/1995) (Sports Sciences Inc.)

The TeeV Golf hardware consists of a wireless (battery-powered) golf club, and a rectangular box which is supposed to be set on the floor. There's a mimmicked (half) golf ball in the middle of the box. According to photos, there are two rows of six "red dots" on the box (these might be nonfunctional decorative elements, or important LEDs/sensors for motion tracking?), some kind of a BIOS cartridge or so (which seems to contain something customized for specific games), and two connection cables (one to the consoles controller port, and one forwarded to a joypad).



```
  |_________:/
```

According to the box: The TeeV SNES version is compatible with PGA Tour Golf (unknown if that refers to the whole PGA series, and unknown if there are other games supported; other games might possibly require other "BIOS" cartridges). The PGA Tour Golf BIOS cartridge does probably translate motion data to a specially timed sequence of joypad button pressed/released signals.
There are TeeV versions for SNES, Sega Genesis, and PC. The US Patent number for the TeeV hardware is 4971325 (with the addition of "other patents pending").

StuntMaster (VictorMaxx)
Advertised as "3-D Virtual Reality Headset".
"Despite what the box says, the StuntMaster VR is not a 3D display. It contains one extremely grainy low resolution LCD screen in the center of the goggles. If you put it on, it hurts your face. The display singes your retinas with an intensely fuzzy, hard-to-focus-on image. The head tracking mechanism is nothing more than a stick you clip to your shoulder (see picture above) which slides through a loop on the side of the headset. When you turn your head, the StuntMaster detects the stick sliding in the loop and translates this into a left or right button press on a control pad, assuming you've actually hooked it up to the controller port of your SNES or Genesis. Remember the "point-of-view instantly scrolls or rotates with the turn of your head" quote? I'd love to see that happen in Super Mario World. Obviously, it couldn't actually work unless the game were programmed for that functionality in advance. Unless, of course, you're playing Doom and you want to turn left or right by moving your head."

```
 1  +6V
 2  GND
 3  Joypad (SNES:DTA, SEGA:Right In)
 4  Joypad (SNES:CLK, SEGA:N/A)
 5  Joypad (SNES:STB, SEGA:Left In)
 6  GND
 7  VCC (SNES:+5V, SEGA:N/A?) (for Joypad?)
 8  N/A
 9  GND
 10 Video in (NTSC composite)
 11 Joypad (SNES:DTA, SEGA:Right Out)
 12 Joypad (SNES:STB, SEGA:Left Out)
 13 GND
 14 Audio in (Left)
 15 Audio in (Right)
Resolution:     240x86 color triads
Field of View:  17 degrees
Weight:         circa 2.5 pounds
```

Nordic Quest (interactive ski-exerciser) (Nordic Track)
The Nordic Quest is an add-on for treadmills (walking-exercising machines) from Nordic Track. Unlike normal treadmills, the Nordic Track one features two handles attached to a string, which the user pulls back and forth during exercising (similar to nordic walking/skiing sticks).
The Nordic Quest includes replacement handles with DPAD (left handle) and buttons (right handle), allowing to play "virtually any" joypad controlled games during exercising; there aren't any special "nordic" games for the controller, instead it can be used for games like golf, car-racing, and flight simulations (as illustrated on the box).
The exercising intensity is claimed to affect the game speed - unknown how this works - maybe by toggling the DPAD on/off, or maybe by toggling the Start (pause) button on/off?

JS-306 Power Pad Tilt (Champ) (joypad with autofire, slowmotion, tilt-mode)
A regular joypad with normal DPAD, the tilt-sensors can be optionally used instead of the DPAD.

Nigal Mouncefill Fly Wheel (Logic 3) (wheel-shaped, tilt-sensor instead dpad)
An odd wheel-shaped controller with tilt-sensors instead of DPAD.

SNES Controllers Lasabirdie (golf club)

The Lasabirdie is a golf club made in 1995 by Ricoh. Supported by only one game (which came shipped with the device):

```
Lasabirdie - Get in the Hole (1995) Ricoh/Good House (JP)
```

Lasabirdie "Golf Mat" and "Golf Club"

```
   ____________________________                 __
   |............................|      insert two     / /
   |:                           :|      AA batteries  / / <-- handle
   |: #      Golf Ball         # :|            |      / /      (rather
   |: #   _ /                   # :|            |     / /       short)
   |: #  (_)       #            # :|            V    / /
   |: #                         # :|          ______/ /
   |: #                         # :|         |       /
  _|:...........................:|           \     / <-- yellow laser symbol
 / |  LT RT   UP DN   A  B   RICOH |           \___/
 | |_____________________________|              |
 / <---- cable (to SNES controller port 2)       |  <-- laser beam
```

The so-called Golf Mat is actually a (not very flat) plastic box, with a mimmicked (half) golf ball mounted on it, the three black fields (shown as ### in the ASCII drawing) might contain laser sensors, the front panel has six buttons: Left/Pause, Right, Up, Down, A/Start and B/Cancel.
For additional/better menu controls, one can connect a normal joypad to SNES port 1.

Below describes the overall transfer protocol. Unknown if/how/what kind of motion, speed, and/or direction information is transmitted via that protocol.

Lasabirdie Controller Data (connected to Port 2)

```
 1st         Button B (CANCEL)    (1=Low=Pressed)
 2nd         Button DOWN          (1=Low=Pressed)
 3rd..6th    Nibble Data bit3-0   (1=Low=One?) (MSB first)
 7th         Nibble Available     (toggles CLK like)
 8th         Packet Available     (1=Low=Yes)
 9th         Button A (START)     (1=Low=Pressed)
 10th        Button UP            (1=Low=Pressed)
 11th        Button LEFT (PAUSE)  (1=Low=Pressed)
 12th        Button RIGHT         (1=Low=Pressed)
 13th..16th  ID Bit3-0       (unknown)  ;read, but not checked by software
 17th..24th  Extra ID Bit7-0 (unknown)  ;read, but not checked by software
 25th and up Unknown/Unused (probably all one, or all zero ?)
```

Command Bytes

Command bytes are used to select a specific packet, and (during the packet transfer) to select nibbles within the previously selected packet:

```
20h        select "GOLF_READY!" ID string packet
22h        select version string packet
30h        select whatever data packet?
3Fh        select whatever data packet?
40h..55h   sent while receiving nibbles number 0..21
5Fh        terminate transfer (or re-select default packet type?)
```

Bytes are output via Port 4201h at a fixed baudrate of circa 10000 bits/second:

```
output 1 start bit ("0")
output 8 data bits (MSB first)
output 2 stop bits ("0","0")
release line (output "1", until the next byte transferred in next frame)
```

Exact time per bit is 2140 master cycles (10036 bps at 21.47727MHz NTSC clock).

Packets

Packets consist of 11 bytes, transferred in 22 nibbles (of 4bit each). For whatever reason, the software receives only one nibble per frame, so a complete packet-transfer takes about 0.36 seconds. The bits are transferred MSB first (bit3,bit2,bit1,bit0), whilst nibbles are transferred LSB first (bit3-0, bit7-4). The 11-byte packets can contain following data:

```
"GOLF_READY!"                ;-ID-string packet  ;\without checksum
FFh,FFh,0,0,0,0,0,0,0,0,0,0  ;-Empty packet      ;/
9 chars, 1 unknown, 1 chksum ;-Version-string    ;\with checksum
10 data bytes, 1 chksum      ;-Normal packet     ;/
```

The checksum (if it is present) is calculated by summing up all 10 data bytes, and adding MSB+LSB of the resulting 16bit sum (ie. sum=sum+sum/100h). The version string packet contains 9 characters (unknown content), one unused byte (unknown value), and the checksum byte. Other packet(s) contain whatever controller/motion data (unknown content).

Below is the procedure for receiving a packet (before doing that, one should first select a packet, eg. send_byte(20h) for receiving the ID string).

```
if [421Ah].bit8 = 0 then exit       ;-exit if no packet available
for i=0 to 21
  old_state = [421Ah].bit9
 @@wait_lop:
  send_byte(40h+i)
  wait_vblank
  if [421Ah].bit9 <> old_state then jmp @@wait_done
  wait_vblank
  if [421Ah].bit9 <> old_state then jmp @@wait_done
  jmp @@wait_lop
 @@wait_done:
  nibble=([421Ah] SHR 10) AND 0Fh
  if (i AND 1)=0 then buf[i/2]=nibble, else buf[i/2]=buf[i/2]+nibble*10h
next i
send_byte(5Fh)               ;-terminate transfer or so
```

SNES Controllers Exertainment (bicycle exercising machine)

The Exertainment is an exercising machine made by Life Fitness. It consists of a stationary bicycle, a monitor with TV tuner, a SNES game cartridge, and a SNES console with some extra hardware plugged into its Expansion Port.

Technical Info

SNES Controllers Exertainment - I/O Ports
SNES Controllers Exertainment - RS232 Controller
SNES Controllers Exertainment - RS232 Data Packets & Configuration
SNES Controllers Exertainment - RS232 Data Packets Login Phase
SNES Controllers Exertainment - RS232 Data Packets Biking Phase

Drawings

SNES Controllers Exertainment - Drawings

Supported Games

```
Cannondale Cup (1993) CEG/American Softworks/RadicalEntertainment (US)
Exertainment Mountain Bike Rally (1994) LifeFitness/RadicalEntertainment (US)
Exertainment Mountain Bike Rally & Speed Racer (combo cart) (1995) (USA)
Exertainment Mountain Bike Rally & Speed Racer (combo cart) (prototype) (EU)
```

Aside from the games, all three Exertainment cartridges are including a "Program Manager", allowing to view/edit user profiles, and, in the old cart from 1994 only - also including some "mini games" called Workout and Fit Test).

Cannondale Cup is essentially same as Mountain Bike Rally, it is sending Exertainment packets (including for checking for the "LIFEFITNES(s)" ID), but it lacks the Program Manager, and even the actual game doesn't seem to react to actions on the exertainment hardware(?).

Playing normal SNES games during exercising isn't supported (the SNES cartridge slot isn't externally accessible, and, selecting the pedal resistance requires special program code in the game cartridge).

The Mountain Bike game works with/without the Exertainment hardware (with the Exertainment features being shown only if the hardware is present).

Joypad Controls

Joypad like controls are attached to the handlebars, featuring the same 12bit button/direction signals as normal joypads. In fact, there should be two such joypads, both mapped as "player 1" input (ie. both wired to Port 4218h). Turning the handlebars isn't possible, instead, steering is done via DPAD Left/Right buttons. Whereas, for the Mountain Bike game, steering is needed only for gaining optional bonus points.

Other Controls

Pedaling speed/force info is probably sent via Port 21C0h Data Packets. The front panel has six buttons - unknown if the buttons states are sent to the SNES - Volume & Program Up/Down and Picture-in-picture are possibly wired directly to the TV set unit, the Menu button is possibly wired to SNES Reset signal(?)

Exertainment Expansion Port Unit - PCB Component List info from byuu

```
U1 40pin TL16C550AN CF62055 N9304 2342265 TI  ;-RS232 controller
U2 20pin PEEL18CV8P CTM22065 333FB    ;-some PAL (sticker "K41A-12802-0000")
U3 20pin 74HC374N (addr.msb & serial) ;\two 8bit latches (13bit SRAM address,
U4 20pin 74HC374N (addr.lsb)          ;/and the 3bit serial-port outputs)
U5 28pin LH5268A-10LL 9348 SHARP      ;-8Kx8 SRAM
U6 16pin ADM232LJN 9403 OF31824       ;-RS232 voltage converter
BATT1    CR2032                       ;-battery (for SRAM)
P2 4pin  short cable to rear 623K-6P4C;-SIO (1=LED?, 2=CLK?, 3=DTA?, 4=/SEL?)
P3 3pin  long cable to front 616M-4P4C;-RS232 (1=GND, 2=N/A, 3=TX, 4=RX)
Px 28pin Connector to SNES expansion port (at bottom of SNES console)
```

SNES Controllers Exertainment - I/O Ports

Exertainment I/O Port Summary (Expansion Port Unit)

```
21C0h.0 TL16C550AN - RX Data FIFO (R)                          ;\when     (?)
21C0h.0 TL16C550AN - TX Data FIFO (W)                          ; DLAB=0   (?)
21C1h.0 TL16C550AN - Interrupt Control (R/W)                   ;/         00h
21C0h.1 TL16C550AN - Baudrate Divisor Latch LSB, Bit0-7 (R/W)  ;\when     (-)
21C1h.1 TL16C550AN - Baudrate Divisor Latch MSB, Bit8-15 (R/W) ;/DLAB=1   (-)
21C2h   TL16C550AN - Interrupt Status (R)                                 01h
21C2h   TL16C550AN - FIFO Control (W)                                     00h
21C3h   TL16C550AN - Character Format Control (R/W)     ;<--- Bit7=DLAB   00h
21C4h   TL16C550AN - Handshaking Control (R/W)                            00h
21C5h   TL16C550AN - RX/TX Status (R) (Write=reserved for testing)        60h
21C6h   TL16C550AN - Handshaking Status (R) (Write=unknown/reserved)      0xh
21C7h   TL16C550AN - Scratch (R/W)                                        (-)
21C8h   74HC374N (U3) - RAM address MSBs and SPI-style Serial Port (W)    (-)
21C9h   Not used
21CAh   74HC374N (U4) - RAM address LSBs (W)                              (-)
21CBh   Not used
21CCh   RAM (U5) data byte to/from selected RAM addr (R/W)   (battery backed)
21CDh   Not used
21CEh   Not used
21CFh   ? initially set to 00h (not changed thereafter) (W) ;\maybe one of
21Dxh   Not used                                            ; these resets
21DFh   ? initially set to 80h (not changed thereafter) (W) ;/the TL16C550AN?
```

21C0h..21C7h - TL16C550AN (U1) (RS232 Controller)

SNES Controllers Exertainment - RS232 Controller
SNES Controllers Exertainment - RS232 Data Packets & Configuration
SNES Controllers Exertainment - RS232 Data Packets Login Phase
SNES Controllers Exertainment - RS232 Data Packets Biking Phase

21C8h - 74HC374N (U3) - RAM address MSBs and SPI-style Serial Port (W)

```
0   Serial Port Select (0=Select, 1=Idle)
1   Serial Port Data   (transferred LSB first)
2   Serial Port Clock  (0=Idle) (data must be stable on 0-to-1 transition)
3-7 Upper 5bit of 13bit RAM address (see Ports 21CAh/21CCh)
```

Used to send two 16bit values (20F3h and 0470h) during initialization (and to send more data later on). This controls some OSD video controller (possibly also the picture-in-picture function). Used values are:

```
20xxh = set address (00h..EFh = yloc*18h+xloc) (24x10 chars)
20Fxh = set address (F0h..F3h = control registers 0..3)
1C20h = ascii space with attr=1Ch ?
1E20h = ascii space with attr=1Eh ?
1Exxh = ascii chars ":" and "0..9" and "A..Z" (standard ascii codes)
1Exxh = lowercase chars "a..z" (at "A..Z"+80h instead of "A..Z"+20h)
0000h = value used for control regs 0 and 1
0111h = value used for control reg 2
0070h,0077h,0470h = values used for control reg 3
```

Unknown if/which bicycle versions are actually using the OSD feature, maybe it has been an optional or unreleased add-on. OSD output is supported in the Program Manager's Workout & Fit Test, apparently NOT for drawing the OSD layer on top of the SNES layer, probably rather for displaying OSD while watching TV programs.
Note: The general purpose "/OUT1" bit (RS232 port 21C4h.Bit2) is also output via the serial port connector (purpose is unknown, might be OSD related, or TV enable, or LED control, or whatever).

21C8h - 74HC374N (U3) - RAM address MSBs and SPI-style Serial Port (W)
21CAh - 74HC374N (U4) - RAM address LSBs (W)
21CCh - SRAM (U5) - RAM data byte to/from selected RAM address (R/W)

```
Port 21CAh.Bit0-7 = RAM Address Bit0-7 (W)   ;\13bit address, 0000h..1FFFh
Port 21C8h.Bit3-7 = RAM Address Bit8-12 (W)  ;/
Port 21C8h.Bit0-2 = See Serial Port description (W)
Port 21CCh.Bit0-7 = RAM Data Bit0-7 (R/W)
```

Used to access battery-backed 8Kbyte SRAM in the expansion port unit. Note: There are additional 2Kbytes of SRAM in the Mountain Bike game cartridge (mapped to 700000h).

21CFh (W) initially set to 00h (not changed thereafter)
21DFh (W) initially set to 80h (not changed thereafter)
Used to configure/reset whatever stuff during initialization (not used thereafter). Maybe one of these ports resets the TL16C550AN?

SNES Controllers Exertainment - RS232 Controller

Texas Instruments TL16C550AN - Asynchronous Communications Element (ACE)
The ACE uses eight I/O addresses (mapped to 21C0h-21C7h in the SNES), the meaning of the first two addresses depends on the "DLAB" bit (which can be changed via 21C3h.Bit7).

21C0h (when DLAB=0) - TL16C550AN - RX Data FIFO (R)

```
0-7  Data (with 16-byte FIFO)
```

21C0h (when DLAB=0) - TL16C550AN - TX Data FIFO (W)

```
0-7  Data (with 16-byte FIFO)
```

21C1h (when DLAB=0) - TL16C550AN - Interrupt Control (R/W)

```
0    Received Data Available Interrupt           (0=Disable, 1=Enable)
1    Transmitter Holding Register Empty Interrupt (0=Disable, 1=Enable)
2    Receiver Line Status Interrupt              (0=Disable, 1=Enable)
3    Modem Status Interrupt                      (0=Disable, 1=Enable)
4-7  Not used (always zero)
```

21C0h (when DLAB=1) - TL16C550AN - Baudrate Divisor Latch LSB, Bit0-7 (R/W)
21C1h (when DLAB=1) - TL16C550AN - Baudrate Divisor Latch MSB, Bit8-15 (R/W)

```
0-7  Divisor Latch LSB/MSB, should be set to "divisor = XIN / (baudrate*16)"
```

21C2h - TL16C550AN - Interrupt Status (R)
```
0    Interrupt Pending Flag (0=Pending, 1=None)
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 21C6h)
01h 3   Transmitter Holding Register Empty   (Ack: Read 21C2h or Write 21C0h)
02h 2   RX FIFO has reached selected trigger level          (Ack: Read 21C0h)
03h 1   RX Overrun/Parity/Framing Error, or Break Interrupt (Ack: Read 21C5h)
06h 2   RX FIFO non-empty & wasn't processed for longer time(Ack: Read 21C0h)
```
Interrupt ID values 04h,05h,07h are not used.

21C2h - TL16C550AN - FIFO Control (W)
```
0    FIFO Enable (0=Disable, 1=Enable) (Enables access to FIFO related bits)
1    Receiver FIFO Reset      (0=No Change, 1=Clear RX FIFO)
2    Transmitter FIFO Reset   (0=No Change, 1=Clear TX FIFO)
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)
```

21C3h - TL16C550AN - Character Format Control (R/W)
```
0-1  Character Word Length    (0..3 = 5,6,7,8 bits)
2    Number of Stop Bits      (0=1bit, 1=2bit; for 5bit chars: only 1.5bit)
3    Parity Enable            (0=None, 1=Enable Parity or 9th data bit)
4-5  Parity Type/9th Data bit (0=Odd, 1=Even, 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)
```

21C4h - TL16C550AN - Handshaking Control (R/W)
```
0    Output Level for /DTR pin  (Data Terminal Ready) (0=High, 1=Low)
1    Output Level for /RTS pin  (Request to Send)     (0=High, 1=Low)
2    Output Level for /OUT1 pin (General Purpose)     (0=High, 1=Low)
3    Output Level for /OUT2 pin (General Purpose)     (0=High, 1=Low)
4    Loopback Mode (0=Normal, 1=Testmode, loopback TX to RX)
5-7  Not used (always zero)
```

21C5h - TL16C550AN - RX/TX Status (R/W, but should accessed as read-only)
```
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).

21C6h - TL16C550AN - Handshaking Status (R/W? - should accessed as read-only)
```
0    Change flag for /CTS pin (Clear to Send)       ;\change flags (0=none,
1    Change flag for /DSR pin (Data Set Ready)      ; 1=changed since last
2    Change flag for /RI pin  (Ring Indicator)      ; read) (automatically
3    Change flag for /DCD pin (Data Carrier Detect) ;/cleared after reading)
4    Input Level on /CTS pin (Clear to Send)        ;\
5    Input Level on /DSR pin (Data Set Ready)       ; current levels
6    Input Level on /RI pin  (Ring Indicator)       ; (inverted ?)
7    Input Level on /DCD pin (Data Carrier Detect)  ;/
```

21C7h - TL16C550AN - Scratch (R/W)
```
0-7  General Purpose Storage (eg. read/write-able for chip detection)
```

Note
The TL16C550AN doesn't seem to support a TX FIFO Full flag, nor automatic RTS/CTS handshaking.

SNES Controllers Exertainment - RS232 Data Packets & Configuration

From Bike to SNES (16 bytes: ATT code, command, 13-byte-data, checksum)
```
Bike Packet 01h ;\these might both contain same bike data,
Bike Packet 02h ;/both required to be send (else SNES hangs)
Bike Packet 03h ;<-- confirms/requests pause mode
Bike Packet 08h Login Part 1 (ID string)
Bike Packet 09h PPU Status Request (with ignored content)
Bike Packet 0Ah Login Part 3 (reply to random values)
Bike Packet 0Ch Login Part 5 (fixed values 00,FF,00,0C,..)
```

From SNES to Bike (13 bytes: ACK code, command, 10-byte-data, checksum)
```
SNES Packet 00h Idle (zerofilled) or PPU Status Response (with "RAD" string)
SNES Packet 01h Biking Start (start biking; probably resets time/distance)
SNES Packet 02h Biking Active (biking)
SNES Packet 03h Biking Pause (pause biking)
SNES Packet 04h Biking Exit (finish or abort biking)
SNES Packet 05h User Parameters
SNES Packet 06h Biking ?
SNES Packet 07h Biking ?
SNES Packet 08h Biking ?
SNES Packet 09h Login Part 2 (random values)
SNES Packet 0Bh Login Part 4 (based on received data)
SNES Packet 0Dh Login Part 6 (login okay)
SNES Packet 0Fh Logout (login failed, or want new login)
SNES Packet 0Ah,0Ch,0Eh (?) (unused?)
```

Packet Details
SNES Controllers Exertainment - RS232 Data Packets Login Phase
SNES Controllers Exertainment - RS232 Data Packets Biking Phase

RS232 Character Format
The character format is initialized as [21C3h]=3Bh, which means,
```
1 start bit, 8 data bits, sticky parity, 1 stop bit, or, in other words:
1 start bit, 9 data bits, no partity, 1 stop bit
```
The sticky parity bit (aka 9th data bit) should be set ONLY in the Bike's ATT characters (133h), all other data (and ACK codes) should have that bit cleared.

RS232 Baudrate
The baudrate is aimed at 9600 bits/sec. The ACE Baudrate Divisor is set to 0023h aka 35 decimal (in both NTSC and PAL versions), with the ACE being driven by the 5.3MHz Dot Clock. The resulting exact timings are:
```
NTSC: 5.36931750MHz/35/16 = 9588.067 Hz
PAL:  5.32034250MHz/35/16 = 9500.612 Hz
```
Notes: The Dot Clock has some slight stuttering on long dots during hblank (but doesn't disturb the baudrate too much). The PAL baudrate doesn't match too well, however, it is the divisor setting closest to 9600 baud.

RS232 Handshaking
The RS232 connector has only 3 pins (RX, TX, GND). The RTS/CTS handshaking signals are thus not used (nor are any Xon/Xoff handshaking characters used). However, there is some sort of handshaking: The "From Bike" packets are preceeded by a ATT (Attention) character (value 133h, with 9th data bit set, aka sticky parity bit set), this allows to resynchronize to packet-start boundaries in case of lost data bytes.
In the other direction, the "From SNES" packets should be sent only in response to successfully received "From Bike" packets.
The packets are small enough to fit into the 16-byte FIFOs of the ACE chip. The baudrate is a bit to low to send 16-byte packets in every frame, so the Bike is apparently pinging out packets at some lower rate.
Note: The SNES software accepts the ATT (Attention) characters only if [21C5h] returns exactly E5h (data present, TX fifo empty, and "error" flags indicating the received partity bit being opposite as normal).

RS232 Interrupts
The ACE Interrupts are left unused: the IRQ pin is probably not connected to SNES, ACE interrupts are disabled via [21C1h]=00h, and ACE interrupt ID in [21C2h] isn't polled by software.

SNES Controllers Exertainment - RS232 Data Packets Login Phase

Login Phase
```
Bike Packet 08h Login Part 1 (ID string)
SNES Packet 09h Login Part 2 (random values)
Bike Packet 0Ah Login Part 3 (reply to random values)
SNES Packet 0Bh Login Part 4 (based on received data)
Bike Packet 0Ch Login Part 5 (fixed values 00,FF,00,0C,..)
SNES Packet 0Dh Login Part 6 (login okay)
(communication phase...)
SNES Packet 0Fh Logout (login failed, or want new login)
```
Login should be done on power-up. And, the SNES software does occassionally logout and re-login (eg. when starting a new game from within main menu).

PPU Status Request
```
Bike Packet 09h PPU Status Request (with ignored content)
SNES Packet 00h PPU Status Response (with "RAD" string)
```
PPU Status can be transferred during Login or Communication Phase, unknown if/when/why the bike is actually doing that (the data is rather useless, except maybe for use as random seed).

From Bike Packet 08h Login Part 1 (ID string)
```
ATT      Attention Code (133h, with 9th bit aka parity set = packet start)
00h      Command (LSB=08h, MSB=Unknown/unused)
01h..0Bh ID String ("LIFEFITNESS" or "LIFEFITNESs") ;[0Bh].bit5=flag? [1EEEh]
0Ch..0Dh Unknown/unused
0Eh      Checksum (00h-[00h..0Dh])
```

From SNES Packet 09h Login Part 2 (random values)
```
ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
00h      Command (LSB=09h, MSB=Zero)
01h..0Ah Random values (RND1,RND2,RND3,RND4,RND5,RND6,RND7,RND8,RND9,RND10)
0Bh      Checksum (00h-[00h..0Ah])
```

From Bike Packet 0Ah Login Part 3 (reply to random values)
```
ATT      Attention Code (133h, with 9th bit aka parity set = packet start)
00h      Command (LSB=0Ah, MSB=Unknown/unused)
01h      RND1+RND5                          ;\
02h      RND2+(RND5*13+9)                   ; RND'values same as in
03h      RND3+((RND5*13+9)*13+9)            ; Login Part 2
04h      RND4+(((RND5*13+9)*13+9)*13+9)     ;
05h      RND5+((((RND5*13+9)*13+9)*13+9)*13+9)  ;/
06h..0Ah Unknown/unused <-- these ARE USED for response TO bike
0Bh..0Dh Unknown/unused <-- these seem to be totally unused
0Eh      Checksum (00h-[00h..0Dh])
```

From SNES Packet 0Bh Login Part 4 (based on received data)
```
ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
00h      Command (LSB=0Bh, MSB=Zero)
01h..0Ah Values "[01h]+[01h..0Ah]" from Login Part 3
0Bh      Checksum (00h-[00h..0Ah])
```

From Bike Packet 0Ch Login Part 5 (fixed values 00,FF,00,0C,..)
```
ATT      Attention Code (133h, with 9th bit aka parity set = packet start)
00h      Command (LSB=0Ch, MSB=Unknown/unused)
01h..0Ah Constants (00h,FFh,00h,0Ch,0Ah,63h,00h,FAh,32h,C8h)
0Bh..0Dh Unknown/unused
0Eh      Checksum (00h-[00h..0Dh])
```

From SNES Packet 0Dh Login Part 6 (login okay)
```
ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
```

```
  00h      Command (LSB=0Dh, MSB=Zero)
  01h..0Ah All zero (00)
  0Bh      Checksum (00h-[00h..0Ah])
```

From SNES Packet 0Fh Logout (login failed, or want new login)

```
  ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
  00h      Command (LSB=0Fh, MSB=Zero)
  01h..0Ah All zero (00)
  0Bh      Checksum (00h-[00h..0Ah])
```

This is sent upon login mismatch, and also if the game wants to re-enter the login phase (as done after leaving the main menu).

From Bike Packet 09h PPU Status Request (with ignored content)

```
  ATT      Attention Code (133h, with 9th bit aka parity set = packet start)
  00h      Command (LSB=09h, MSB=Unknown/unused)
  01h..0Dh Unknown/unused
  0Eh      Checksum (00h-[00h..0Dh])
```

From SNES Packet 00h PPU Status Response (with "RAD" string)

```
  ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
  00h      Command (LSB=00h, MSB=Zero)
  01h      PPU2 Status   [213Fh] (PPU2 chip version & Interlace/Lightgun/NTSC)
  02h      PPU1 Status   [213Eh] (PPU1 chip version & OBJ overflow flags)
  03h      CPU  Status   [4210h] (CPU chip version & NMI flag)
  04h      Curr Scanline [213Dh] (lower 8bit of current scanline number)
  05h..0Ah Constants (52h,41h,44h,00h,00h,00h) (aka "RAD",0,0,0)
  0Bh      Checksum (00h-[00h..0Ah])
```

Note: This data is send only after PPU Status Request. There are also cases (during menues) where SNES is sending Packet 00h with zerofilled data body.

SNES Controllers Exertainment - RS232 Data Packets Biking Phase

Communication Phase

```
  SNES Packet 00h Idle (while in menu) (also used for PPU Status Response)
  SNES Packet 01h Biking Start (start biking; probably resets time/distance)
  SNES Packet 02h Biking Active (biking)
  SNES Packet 03h Biking Pause (pause biking)
  SNES Packet 04h Biking Exit (finish or abort biking)
  SNES Packet 05h User Parameters
  SNES Packet 06h Biking ?
  SNES Packet 07h Biking ?
  SNES Packet 08h Biking ?
  Bike Packet 01h ;\these might both contain same bike data,
  Bike Packet 02h ;/both required to be send (else SNES hangs)
  Bike Packet 03h ;<-- confirms/requests pause mode
```

Unknown values & commands might include things like TV control.

From Bike Packet xxh (Communication Phase)

```
  ATT      Attention Code (133h, with 9th bit aka parity set = packet start)
  00h      Command (LSB=00h..07h or so, MSB=Curr Level 0..12, Pedal Resistance)
  01h      Speed in pedal rotations per minute (0..xxx) (above 200=glitches?)
  02h      Time (MSB) in 60 second units (0..255)       ;\
  03h      Time (LSB) in 1 second units (0..59)         ;/
  04h      Calories per hour (MSB) in 256 cal/hr units  ;\this used in bank B0h
  05h      Calories per hour (LSB) in 1 cal/hr units    ;/
  06h      Calories burned (MSB) in 256/4 cal units     ;\
  07h      Calories burned (LSB) in 1/4 cal units       ;/
  08h      Distance (MSB) in 65536/3600 miles           ;\
  09h      Distance (MID) in 256/3600 miles             ;
  0Ah      Distance (LSB) in 1/3600 miles               ;/
  0Bh      Pulse in heart beats per minute (1..255 bpm, or 0=No Pulse Sensor)
  0Ch      Fit Test Score (clipped to range 10..60)
  0Dh      Whatever 8bit (invalid values CRASH combo-cart games)
  0Eh      Checksum (00h-[00h..0Dh])
```

In Fit Test mode (when [0Dh]=85h), the bike seems to send Time=Zero when the test ends (after 5 minutes) - either it's counting time backwards in that mode, or it's wrapping from 5 minutes to zero? Other modes, like workout, are counting time upwards, and end when reaching the selected time goal value.

From SNES Packet xxh

```
  ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
  00h      Command (LSB=01h..0xh ?, MSB=Wanted Level 0..12 Pedal Resistance)
  01h      Something (MSB=often same as above MSB, LSB=Present Hill related)
  02h      Present Hill
  03h..09h Upcoming Hills (7 bytes)
  0Ah      Whatever (in MENU: 01h,02h,00h, WORKOUT:80h, FIT-TEST:85h) 80h..86h
  0Bh      Checksum (00h-[00h..0Ah])
```

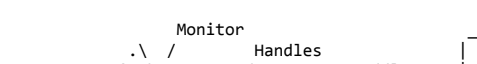
From SNES Packet x5h (User Parameters)

```
  ACK      Acknowledge Code (33h, received packet with good checksum From Bike)
  00h      Command (LSB=05h, MSB=Wanted Level 0..12, Pedal Resistance)
  01h      Player's Sex (00h=Female, 01h=Male)
  02h      Player's Age in years (0..99)
  03h      Player's Weight in pounds (0..255, plus below Weight Extra)
  04h      Player's Pulse in heart beats per minute
  05h      Player's Weight Extra (added to weight, eg. 399 = values FFh+90h)
  06h..09h Garbage, set to same value as [05h], but NOT counted in checksum?
  0Ah      Whatever (same as in other "From SNES" communication packets)
  0Bh      Checksum (00h-[00h..0Ah]) --- in this case, excluding [06h..09h] ?
```

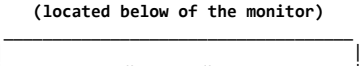
Single-cart is configuring this packet for Fit Test (but is never sending the packet)? Combo-cart is often sending this packet (but with all parameters set to zero)?

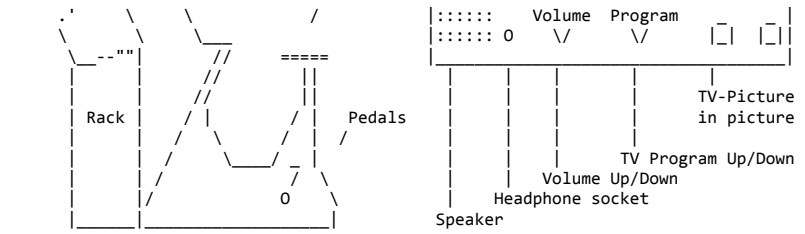
SNES Controllers Exertainment - Drawings

Exercising Machine (Side View)

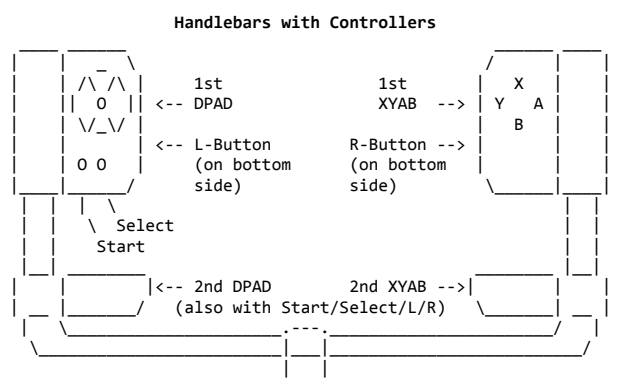


Front Panel
(located below of the monitor)





The monitor, rack and front panel have been spotted on prototype photos, unknown if the normal retail bikes did have them, too (also possible that one got only the bike and expansion port unit - and had to use it with a regular SNES and TV-set).



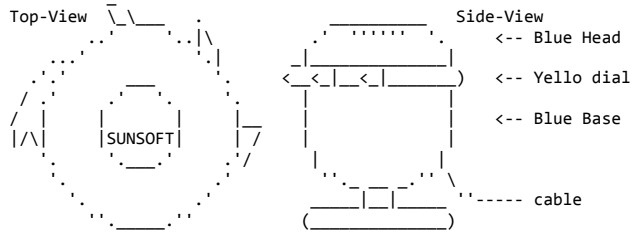
As depicted, there appear to be absolutely no brakes on this bike. Which must have made it a frightening kamikaze-like experience to sit on that thing. Even worse, there is no bell. One could only scream "Out of the way! Out of the way!" at the monitor.

SNES Controllers Pachinko

Pachinko Controller (Sunsoft)
The Pachinko controller should be connected to controller Port 2 (plus a normal joypad in Port 1 for menu selections). After the usual joypad strobing, Pachinko data can be read serially from [4017h].Bit0:

```
1st..8th    Unknown/unused               (would be Buttons/DPAD on joypads)
9th         Used... probably a button?   (would be Button-A on joypads)
10th..12th  Unknown/unused               (would be Buttons on joypads)
13th..16th  ID Bit3-0       (must be 0Eh) (MSB first)
17th..24th  Extra ID Bit7-0 (must be 77h) (MSB first)
25th        Unknown/unused
26th..32th  Analog Dial Position (7bit, MSB first, inverted, 1=Low=Zero)
33th..      Unknown/padding
```

Average analog 7bit range returned on real hardware is unknown. In software, the used 7bit range is around 18h=Stopped through 7Fh=Fastest.
The controller looks somewhat like a blue egg, plus a yellow dial with zagged finger-grips, and probably with a button somewhere (maybe the orange window in the middle of the head or so?):



Known supported games are:

```
Hissatsu Pachinko Collection 1 (J) 1994 Sunsoft/Fuji
Hissatsu Pachinko Collection 2 (J) 1995 Sunsoft/Fuji
Hissatsu Pachinko Collection 3 (J) 1995 Sunsoft/Daiichi/Nifty-Serve
Hissatsu Pachinko Collection 4 (J) 1996 Sunsoft/Kyoraku/Nifty-Serve
```

Note: Pachinko is a japanese gambling game; its appearance is resembling pinball, but concerning stupidity it's more resembling one-armed-bandit-style slot machines.

SNES Controllers Other Inputs

Reset Button (on the console)
Resets various CPU/PPU/APU registers, but leaves WRAM intact, so for example, games could use the button to restart the current level, or to enter the main menu.
Note: The three SPC7110 games are containing a selftest program (executed upon first boot; when battery-backed SRAM is still unitialized), the user is prompted to push the Reset Button after each test screen.

Super Famicom Box
This thing has a (external) coin input, two push buttons, and a 6-position switch (all these inputs are accessible only by its HD64180 CPU though, not by the SNES CPU).
The two attached joypads aren't directly wired to the SNES, instead, they are first passed to the HD64180, and then forwarded to SNES via 16bit shift registers. This allows the HD64180 to sense the "L+R+Select+Start" Soft-Reset key combination, and to inject recorded controller data in Demo/Preview mode. On the restrictive side, the 16bit shift registers are making it incompatible with special controllers like mice (which won't work with most of the SFC-Box games anyways); unless, maybe "automatic-reading" mode bypasses the 16bit shift-register, and forwards the FULL bitstream directly from SFC-Box to SNES? Moreover, during Game Selection, some normally unused bits of the WRIO/RDIO "controller" port are misused to communicate between KROM1 (on HD64180) and ATROM (on SNES).

SNES Add-On Turbo File (external backup memory for storing game positions)

The Turbo File add-ons are an external battery-backed RAM-Disks made by ASCII. Turbo File hardware has been produced for NES, SNES, 8bit Gameboy, and Gameboy Advance. It's been sold only in japan, and it's mainly supported by ASCII's own games. The SNES related hardware versions are:

```
SNES Turbo File Twin (160K) (128K in STF mode, 4x8K in TFII mode)
SNES Turbo File Adapter (SNES adapter for NES Turbo File & Turbo File II)
```

TFII Mode (old NES mode, 4x8Kbyte)
SNES Add-On Turbo File - TFII Mode Transmission Protocol
SNES Add-On Turbo File - TFII Mode Filesystem

STF Mode (native SNES mode, 128Kbyte)
SNES Add-On Turbo File - STF Mode Transmission Protocol
SNES Add-On Turbo File - STF Mode Filesystem

Compatible Games
SNES Add-On Turbo File - Games

NES Turbofile (AS-TF02)
Original NES version, contains 8Kbytes battery backed RAM, and a 2-position PROTECT switch, plus a LED (unknown purpose).

NES Turbo File II (TFII)
Newer NES version, same as above, but contains 32Kbytes RAM, divided into four 8Kbyte slots, which can be selected with a 4-position SELECT switch.

SNES Turbo File Adapter
Allows to connect a Turbo File or Turbo File II to SNES consoles. Aside from the pin conversion (15pin NES to 7pin SNES), it does additionally contain some electronics (for generating a SNES controller ID, and a more complicated protocol for entering the data-transfer phase). Aside from storing SNES game positions, this can be also used to import NES files to SNES games.

SNES Turbo File Twin
SNES version with 160Kbyte SRAM, and with 5-position mode SELECT switch. 128K used in STF mode ("SNES Super Turbo File"), and 4x8K used in TFII modes 1/2/3/4 (equivalent to NES Turbo File II with SNES Turbo File Adapter).

Small square box that connects via cable to controller port.
Two position PROTECT switch (off/on)
Five position SELECT switch (STF, and "TFII" 1,2,3,4)
There is a red LED. And two 1.5V batteries?

Hardware Versions

```
Name               Capacity                Connection
Turbofile (AS-TF02) 1x8Kbyte               NES-to-SNES Adapter
Turbo File II      4x8Kbyte                NES-to-SNES Adapter
Turbo File Twin    4x8Kbyte plus 128Kbyte  Direct SNES Connection
Gameboy version    ?                       N/A ?
Gameboy Advance    ?                       N/A ?
```

SNES Add-On Turbo File - TFII Mode Transmission Protocol

oldest_recv_8191_bytes:

```
call oldest_invoke_transfer   ;start transfer
if invoke_okay then for i=0001h to 1FFFh, oldest_recv_byte(buf[i])
jmp oldest_reset_turbofile    ;end transfer (always, even if invoke failed)
```

oldest_send_8191_bytes:

```
call oldest_invoke_transfer   ;start transfer
if invoke_okay then for i=0001h to 1FFFh, oldest_send_byte(buf[i])
jmp oldest_reset_turbofile    ;end transfer (always, even if invoke failed)
```

oldest_invoke_transfer:

```
call oldest_detect_and_get_status
if no_tf_connected then fail/exit
if data_phase=1 then            ;oops, already invoked
  oldest_detect_and_get_status  ;abort old transfer
  jmp oldest_invoke_transfer    ;retry invoking new transfer
[004016]=01h                    ;strobe on      ;\
for i=1 to 15,dummy=[004017],next i  ;issue 15 clks  ; invoke transfer
[004016]=00h                    ;strobe off     ; (16 clks total)
dummy=[004017]                  ;issue 1 clk    ;/
call oldest_detect_and_get_status                   ;\want flag set now
if data_phase=0 then fail/exit                      ;/
for i=1 to 7                                        ;\skip remaining 7 bits
  dummy=[004017]  ;<-- required?    ;issue clk      ; of unused byte 0000h
  [004016]=01h                  ;strobe on      ; (the first bit was
  [004016]=00h                  ;strobe off     ; skipped by STROBE in
next i                                              ;/detect_and_get_status)
```

After above, the hardware byte-address is 0001h (ie. unlike as in NES version, the unused byte at address 0000h is already skipped).

oldest_detect_and_get_status:

```
[004016]=01h    ;strobe on
[004016]=00h    ;strobe off
for i=23 to 0   ;get ID/status (MSB first)
  temp=[004017]  ;issue clk & get data
  stat.bit(i)=temp.bit0
next i
if stat.bit(11..8)<>0Eh then no_tf_connected=1  ;major 4bit id
if stat.bit(7..0)<>0FFh then no_tf_connected=1  ;minor 8bit id
if stat.bit(12)=1 then data_phase=1 else data_phase=0
```

oldest_reset_turbofile:
```
[004016]=01h    ;strobe on
dummy=[004017]  ;issue clk
[004016]=00h    ;strobe off
dummy=[004017]  ;issue clk
[004016]=00h    ;strobe off (again?)
jmp oldest_detect_and_get_status
```

oldest_recv_byte(data):
```
for i=0 to 7  ;transfer data byte (LSB first)
  temp=[004017]               ;issue clk (required?), and get bit from joy4
  data.bit(i)=temp.bit(1)     ;extract received data bit
  [004016]=01h                ;strobe on
  [004201]=80h*data.bit(i)    ;write SAME/UNCHANGED bit to hardware
  [004016]=00h                ;strobe off (WRITE CLOCK)
next i
```

oldest_send_byte(data):
```
for i=0 to 7  ;transfer data byte (LSB first)
  dummy=[004017]              ;issue clk (really required for writing?)
  [004016]=01h                ;strobe on
  [004201]=80h*data.bit(i)    ;write NEW bit to hardware
  [004016]=00h                ;strobe off (WRITE CLOCK)
next i
```

SNES Add-On Turbo File - TFII Mode Filesystem

Turbo File Memory
The first byte (at offset 0000h) is unused (possibly because that there is a risk that other games with other controller access functions may destroy it); after resetting the address, one should read one dummy byte to skip the unused byte. The used portion is 8191 bytes (offset 0001h..1FFFh). The "filesystem" is very simple: Each file is attached after the previous file, an invalid file ID indicates begin of free memory.

Turbo File Fileformat (newer files) (1987 and up)
Normal files are formatted like so:
```
2   ID "AB" (41h,42h)
2   Filesize (16+N+2) (including title and checksum)
16  Title in ASCII (terminated by 00h or 01h)
N   Data Portion
2   Checksum (all N bytes in Data Portion added together)
```

Turbo File Fileformat (old version) (1986)
The oldest Turbo File game (NES Castle Excellent from 1986) doesn't use the above format. Instead, it uses the following format, without filename, and with hardcoded memory offset 0001h..01FFh (511 bytes):
```
1   Don't care (should be 00h)    ;fixed, at offset 0001h
2   ID AAh,55h                    ;fixed, at offset 0002h..0003h
508 Data Portion (Data, end code "BEDEUTUN", followed by some unused bytes)
```
CAUTION:
```
The early version has transferred all bytes in reversed bit-order,
so above ID bytes AAh,55h will be seen as 55h,AAh in newer versions!
```
Since the address is hardcoded, Castle Excellent will forcefully destroy any other/newer files that are located at the same address. Most newer NES/SNES games (like NES Fleet Commander from 1988, and SNES Wizardry 5 from 1992) do include support for handling the Castle Excellent file. One exception that doesn't support the file is NES Derby Stallion - Zenkoku Ban from 1992.

SNES Add-On Turbo File - STF Mode Transmission Protocol

FileTwinSendCommand (28bits)
```
set strobe=1
8x sendbit (LSB first)    ;command (24h=read or 75h=write)
20x sendbit (LSB first)   ;address (00000h..FFFFFh)
set strobe=0
FileTwinRecvStatusAndID
error if bad-ID or general-error-flag (for write: also write protect-error)
retry "FileTwinSendCommand" if desired Read (or Write) Mode bit isn't set
```
Thereafter, send/receive data byte(s), and finish by TerminateCommand

TerminateCommand
```
set strobe=1
if command was READ then issue clk (don't do that on WRITE command)
set strobe=0
FileTwinRecvStatusAndID
retry "TerminateCommand" if Data Read/Write Mode bits are still nonzero
```

FileTwinSendDataByte
```
set strobe=1
8x sendbit (LSB first)
set strobe=0
```

FileTwinRecvDataByte
```
set strobe=1
set strobe=0
8x recvbit (from joy4) (LSB first) (inverted)
```

FileTwinRecvStatusAndID (32bits)
```
set strobe=1
set strobe=0
12x recvbit (from joy2) (ignored)
4x  recvbit (from joy2) (MSB first) (major ID, must be 0Eh)
8x  recvbit (from joy2) (MSB first) (minor ID, must be FEh)
1x  recvbit (from joy2) Data Write Mode (0=No/Idle, 1=Yes/Command 75h)
1x  recvbit (from joy2) Data Read Mode  (0=No/Idle, 1=Yes/Command 24h)
1x  recvbit (from joy2) General-Hardware-Error (1=Error)
1x  recvbit (from joy2) Write-Protect-Error    (1=Error/Protected)
```

```
4x  recvbit (from joy2) (MSB first) (capacity) (usually/always 0=128K)
```

Low Level Functions

```
set strobe=1        --> [004016h]=1
set strobe=0        --> [004016h]=0
recvbit (from joy2) --> bit=[004017h].bit0
recvbit (from joy4) --> bit=NOT [004017h].bit1
sendbit             --> [004201h]=bit*80h, dummy=[004017h]
issue clk           --> dummy=[004017h]
```

SNES Add-On Turbo File - STF Mode Filesystem

FileTwinCapacityCodes (last 4bit of FileTwinRecvStatusAndID)

```
00h  Single Drive with 128Kbytes (normal) (plus extra 32Kbyte for TFII mode)
01h  Single Drive with 256Kbytes
02h  Single Drive with 384Kbytes
03h  Single Drive with 640Kbytes (really, this is NOT 512Kbytes)
04h  Multi-Drive with 1 normal 128K Drive  (128K total) ;\allows to READ from
05h  Multi-Drive with 2 normal 128K Drives (256K total) ; all drives, but
06h  Multi-Drive with 3 normal 128K Drives (384K total) ; can WRITE only
07h  Multi-Drive with 5 normal 128K Drives (640K total) ;/to first drive
08h..0Fh  Reserved (treated same as 00h; Single Drive with 128Kbytes)
```

FileTwinAddresses (?)

XXX multiply below by 400h

```
000h..00Fh  FAT (4Kbytes)
010h..1FFh  Entries for 1st 124Kbytes
200h..7FFh  Unused
800h..9FFh  Entries for 2nd 128Kbytes (if any)
A00h..BFFh  Entries for 3rd 128Kbytes (if any)
C00h..DFFh  Entries for 4th 128Kbytes (if any) (seems to be bugged)
E00h..FFFh  Unused
xxxh..xxxh  Partition-Read FAT (though WRITES are to address zero ?)
```

FileTwinFAT

The FAT is 4096 bytes in size:

```
000h..1EFh  Entries for 1st 124Kbytes
1F0h..3EFh  Entries for 2nd 128Kbytes (if any)
3F0h..5EFh  Entries for 3rd 128Kbytes (if any)
5F0h..7EFh  Entries for 4th 128Kbytes (if any) (seems to be bugged)
7F0h..FFBh  Unused
FFCh..FFFh  ID "FAT0"
```

Each FAT Entry is 32bit (4 bytes) wide:

```
0-11   filesize in kbyte (1st blk), 000h (2nd..Nth blk), FFFh (free blk)
12-23  NNNh (next blk), FFFh (last blk), or also FFFh (free blk)
24-31  8bit sector chksum (all 1024 bytes added together), or FFh (free blk)
```

Note: The above FFFh values should be as so (though older games are checking only the upper 4bit, thereby treating any Fxxh values as free/last block). Unused FAT entries (that exceed memory capacity) are also marked as "free".

FileTwinFileHeaders

The first 24 bytes (in the first sector) of a file contain File ID & Name:

```
000h..003h ID1 (must be "SHVC")
004h..007h ID2 (should be same as Game Code at [FFB2h] in ROM-Header)
008h..017h Filename (padded with ...spaces ?)
018h..     File Data (Filesize from FAT, multiplied by 1024, minus 24 bytes)
```

ID2 and Name may contain ASCII characters 20h..3Fh and 41h..5Ah.

SNES Add-On Turbo File - Games

SNES Games that support Turbo File Twin in STF-Mode

```
Bahamut Lagoon (1996) Square (INCORRECTLY?! claimed to support Turbo File)
Daisenryaku Expert WWII: War in Europe (1996) SystemSoftAlpha/ASCII Corp (JP)
Dark Law - Meaning of Death (1997) ASCII (JP)
Derby Stallion III (1995) (supports both TFII and STF modes)
Derby Stallion 96 (1996) (supports TFII and STF and Satellaview-FLASH-cards)
Derby Stallion 98 (NP) (1998) (supports both TFII and STF modes)
Gunple/Ganpuru - Gunman's Proof (1997) ASCII/Lenar (JP)
Mini Yonku/4WD Shining Scorpion - Let's & Go!! (1996) KID/ASCII Corp (JP)
Ongaku Tukool/Tsukuru Kanaderu (supports STF and Satellaview-FLASH-cards)
RPG Tukool 1
RPG Tukool 2 (supports STF and Satellaview-FLASH-cards)
Sound Novel Tukool (supports STF and Satellaview-FLASH-cards)
Tactics Ogre - Let Us Cling Together (supports both TFII and STF modes)
Wizardry 6 - Bane of the Cosmic Forge (1995) (JP) (English)
```

Plus (unverified):

```
STF: Tower Dream
STF: Best Shot Pro Golf (J)
STF: Jewel of Live (RPG Maker Super Dante) (BS)
STF: Solid Runner (J)
STF: Wizardry Gaiden IV - Taima no Kodou (J) (v1.1)
```

SNES Games that support Turbo File Adapter & Turbo File Twin in TFII-Mode

```
Ardy Lightfoot (1993)
Derby Stallion II (1994)
Derby Stallion III (1995) (supports both TFII and STF modes)
Derby Stallion 96 (1996) (supports TFII and STF and Satellaview-FLASH-cards)
Derby Stallion 98 (NP) (1998) (supports both TFII and STF modes)
Down the World: Mervil's Ambition (1994)
Kakinoki Shogi (1995) ASCII Corporation
Tactics Ogre - Let Us Cling Together (supports both TFII and STF modes)
Wizardry 5 - Heart of the Maelstrom (1992) Game Studio/ASCII (JP)
BS Wizardry 5 (JP) (Satellaview BS-X version)
```

Plus (unverified):

```
TFII: Haisei Mahjong - Ryouga (J)
TFII: Super Fire Pro Wrestling Special, X, and X Premium (J)
TFII: Super Robot Taisen Gaiden - Masou Kishin - The Lord of Elemental (J)
```

Note: The US version of Wizardry 5 (1993) contains 99% of the turbo file functions, but lacks one opcode that makes the hardware detection nonfunctional. Wizardry 5 was announced/rumoured to be able to import game positions from NES to SNES.

NES Games that do support the Turbo File / Turbo File II

```
Best Play Pro Yakyuu (1988) ASCII (J)
Best Play Pro Yakyuu '90 (1990) (J)
Best Play Pro Yakyuu II (1990) (J)
Best Play Pro Yakyuu Special (1992) (J)
Castle Excellent (1986) ASCII (J) (early access method without filename)
Derby Stallion - Zenkoku Ban (1992) Sonobe Hiroyuki/ASCII (J)
Downtown - Nekketsu Monogatari (19xx) Technos Japan Corp (J)
Dungeon Kid (1990) Quest/Pixel (J)
Fleet Commander (1988) ASCII (J)
Haja no Fuuin (19xx) ASCII/KGD (J)
Itadaki Street - Watashi no Mise ni Yottette (1990) ASCII (J)
Ninjara Hoi! (J)
Wizardry - Legacy of Llylgamyn (19xx?) (J)
Wizardry - Proving Grounds of the Mad Overlord (1987) (J)
Wizardry - The Knight of Diamonds (1991) (J)
```

NES games that do support Turbo File should have a "TF" logo on the cartridge.
Note: Castlequest (US version of Castle Excellent) reportedly lacks support.

SNES Add-On Barcode Battler (barcode reader)

The Barcode Battler from Epoch allows to scan barcodes (either from special paper cards, or from daily-life products like food packagings), games can then use the barcode digits as Health Points, or other game attributes.

Standalone-Mode
The device was originally designed as stand-alone gaming console with some push buttons, a very simple LCD screen with 7-segment digits & some predefined LCD symbols, and a built-in game BIOS (ie. without external cartridge slot, and without any bitmap graphics).

Link-Mode
Later versions (with black case) include an "EXT" link port, allowing to link to other Barcode Battler hardware, or to Famicom/Super Famicom consoles. The EXT port is probably bi-directional, but existing Famicom/Super Famicom games seem to be using it only for reading barcodes (without accessing the LCD screen, push buttons, speaker, or EEPROM).

SNES Add-On Barcode Transmission I/O
SNES Add-On Barcode Battler Drawings

Barcode Battler Famicom (NES) Games

```
Barcode World (1992) Sunsoft (JP) (includes cable with 15pin connector)
```

Barcode Battler Super Famicom (SNES) Games

```
Alice's Paint Adventure (1995)
Amazing Spider-Man, The - Lethal Foes (19xx)
Barcode Battler Senki Coveni Wars (1993) Epoch
Donald Duck no Mahou no Boushi (19xx)
Doraemon 2: Nobita's Great Adventure Toys Land (1993)
Doraemon 3: Nobita and the Jewel of Time (1994)
Doraemon 4 - Nobita to Tsuki no Oukoku (19xx)
Doroman (canceled)
Dragon Slayer - Legend of Heroes 2 (1993) Epoch
J-League Excite Stage '94 (1994)
J-League Excite Stage '95 (1995)
Lupin Sansei - Densetsu no Hihou wo Oe! (19xx)
Super Warrior Combat (19xx - does this game exist at all?)
```

Barcode Battler Hardware Versions

Region	Case	EXT	Barcode-Reader	Name	Year
Japan	White	None	Yes	Barcode Battler	1991
Japan	Black	1	Yes	Barcode Battler II	1992
Japan	Black	2	None	Barcode Battler II^2	199x
Europe	Black	1	Yes	Barcode Battler	1992/1993

The versions with one EXT socket can be connected to NES/SNES, or to one or more of the "II^2" units (allowing more players to join the game).

Connection to SNES/NES consoles
Connection to Super Famicom or SNES requires a "BBII INTERFACE": a small box with 4 LEDs and two cables attached (with 3pin/7pin connectors), the interface has been sold separetedly, it's needed to add a SNES controller ID code to the transmission protocol.
Connection to Famicom consoles requires a simple cable (without interface box) (with 3pin/15pin connectors), the cable was shipped with the "Barcode World" Famicom cartridge, connection to NES would require to replace the 15pin Famicom connector by 7pin NES connector.
The required 3pin EXT connector is available only on newer Barcode Battlers (with black case), not on the original Barcode Battler (with white case).

```
Unknown if all 3 pins are actually used by NES/SNES cable/interface?
Unknown if NES/SNES software can access LCD/buttons/speaker/EEPROM ?
```

Connectivity
"Connectivity mode is accessible if you plug in a standard 3.5mm mono jack plug into the expansion port on the left hand side of the unit, hold down the R-Battle and R-Power buttons and turn the unit on, the Barcode Battler II goes into scanner mode."

Barcode Battler II Interface
The hardware itself was manufactured by Epoch, and licensed by Nintendo (it says so on the case).
The four lights, from left to right, indicate as follows:

```
"OK"    All is well, the device is operating as normal.
"ER"    Maybe there's something wrong?
"BBII"  The Barcode Battler is sending data to the device.
"SFC"   The SFC/SNES is waiting for a signal from the Barcode Battler.
```

Component List (may be incomplete)

```
80pin NEC uPD75316GF (4bit CPU with on-chip 8Kx8 ROM, 512x4 RAM, LCD driver)
```

```
8pin Seiko S2929A (Serial EEPROM, 128x16 = 2Kbit) (same/similar as S29290)
3pin EXT socket (3.5mm "stereo" jack) (only in new versions with black case)
LCD Screen (with 7-segment digits and some predefined words/symbols)
Five LEDs (labelled "L/R-Battle Side")
Seven Push Buttons (L/R-POWER, L/R-Battle, Power on/off, Select, Set)
Speaker with sound on/off switch (both on bottom side)
Barcode reader (requires card-edges to be pulled through a slot)
Batteries (four 1.5V AA batteries) (6V)
```

SNES Add-On Barcode Transmission I/O

The Barcode Battler outputs barcodes as 20-byte ASCII string, at 1200 Baud, 8N1. The NES software receives that bitstream via Port 4017h.Bit2. The SNES software requires a BBII Interface, which converts the 8bit ASCII digits into 4bit nibbles, and inserts SNES controller ID and status codes, the interface should be usually connected to Controller Port 2 (although the existing SNES games seem to accept it also in Port 1).

Barcode Battler (with BBII Interface) SNES Controller Bits

```
1st..12th   Unknown/unused (probably always 0=High?)
13th..16th  ID Bits3..0          (MSB first, 1=Low=One) (must be 0Eh)
17th..24th  Extended ID Bits7..0 (MSB first, 1=Low=One) (must be 00h..03h)
            (the SNES programs accept extended IDs 00h..03h, unknown
            if/when/why the BBII hardware does that send FOUR values)
25th        Status: Barcode present (1=Low=Yes)
26th        Status: Error Flag 1 ?
27th        Status: Error Flag 2 ?
28th        Status: Unknown      ?
```

Following bits need/should be read ONLY if the "Barcode Present" bit is set.

```
29th-32th   1st Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
33th-36th   2nd Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
37th-40th   3rd Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
41th-44th   4th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
45th-48th   5th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
49th-52th   6th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
53th-56th   7th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
57th-60th   8th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
61th-64th   9th Barcode Digit, Bits3..0  (MSB first, 1=Low=One)
65th-68th   10th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
69th-72th   11th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
73th-76th   12th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
77th-80th   13th Barcode Digit, Bits3..0 (MSB first, 1=Low=One)
81th and up Unknown/unused
     Above would be 13-digit EAN-13 codes
     Unknown how 12-digit UPC-A codes are transferred   ;\whatever leading
     Unknown if/how 8-digit EAN-8 codes are transferred ; or ending padding?
     Unknown if/how 8-digit UPC-E codes are transferred ;/
```

For some reason, delays should be inserted after each 8 bits (starting with 24th bit, ie. after 24th, 32th, 40th, 48th, 56th, 64th, 72th bit, and maybe also after 80th bit). Unknown if delays are also needed after 8th and 16th bit (automatic joypad reading does probably imply suitable delays, but errors might occur when reading the ID bits via faster manual reading).

Barcode Battler RAW Data Output

Data is send as 20-byte ASCII string. Bytes are transferred at 1200 Bauds:

```
1 Start bit (must be LOW)
8 Data bits (LSB first, LOW=Zero, HIGH=One)
1 Stop bit  (must be HIGH)
```

The first 13 bytes can contain following strings:

```
"nnnnnnnnnnnn"    ;13-digit EAN-13 code (ASCII chars 30h..39h)
<Unknown>         ;12-digit UPC-A code (with ending/leading padding?)
"     nnnnnnn"    ;8-digit EAN-8 code (with leading SPC-padding, ASCII 20h)
<Unknown>         ;8-digit UPC-E code (with ending/leading padding?)
"ERROR        "   ;indicates scanning error
```

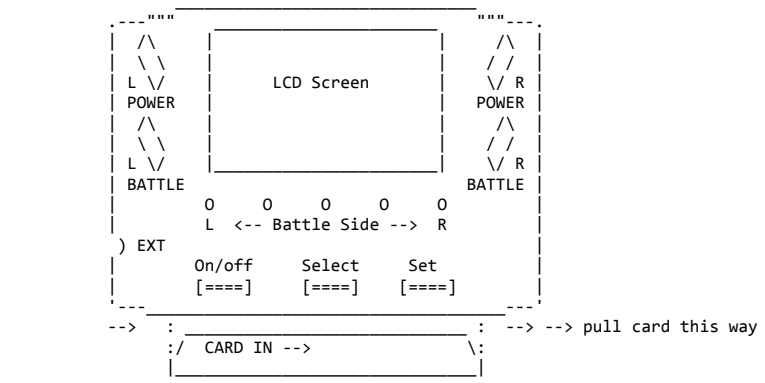
The last 7 bytes must contain either one of following ID strings:

```
"EPOCH",0Dh,0Ah   ;<-- this is sent/accepted by existing hardware/software
"SUNSOFT"         ;<-- this would be alternately accepted by the NES game
```

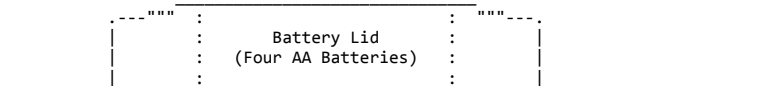
There are rumours that one "must" use a mono 3.5mm plug in order to receive data - that's obviously bullshit, but it might indicate that the middle pin of stereo plugs must be GNDed in order to switch the Barcode Battler into transmit mode(?)

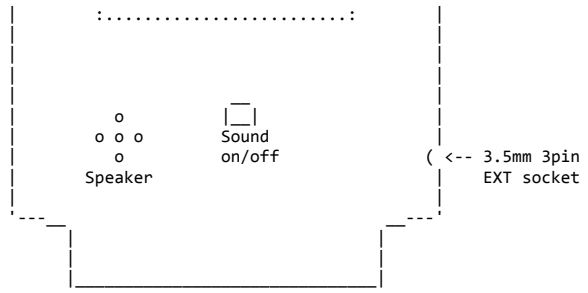
SNES Add-On Barcode Battler Drawings

Barcode Battler - Handheld Console (Front)

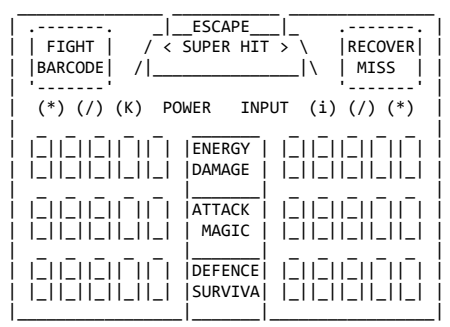


Barcode Battler - Handheld Console (Back)

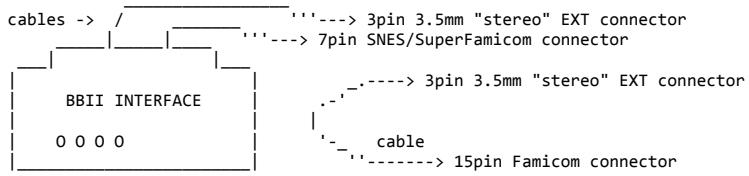




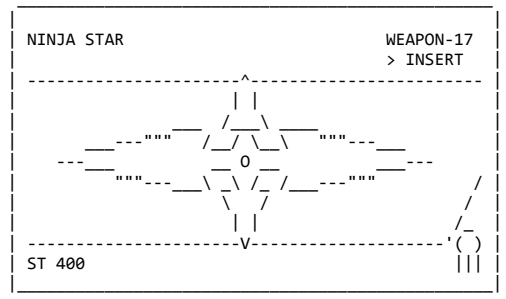
Barcode Battler - LCD Screen Layout



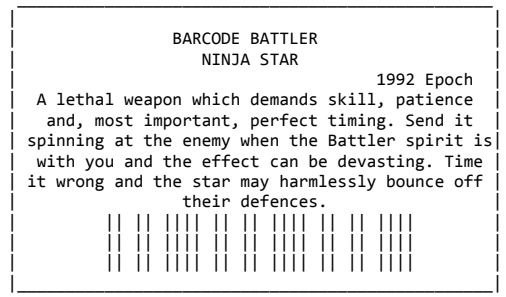
Barcode Batter II Interface (SNES/SuperFamicom) & Simple Cable (Famicom)



Paper-Card Front (Picture Side)



Paper-Card Back (Description & Barcode)



SNES Add-On SFC Modem (for JRA PAT)

SNES Add-On SFC Modem - Data I/O
SNES Add-On SFC Modem - Misc

Controller
The SFC Modem is bundled with a special controller (to be connected to controller port 1) (required for the JRA PAT software):
SNES Controllers NTT Data Pad (joypad with numeric keypad)

FLASH Backup
The JRA PAT Modem BIOS cartridges contain FLASH backup memory (unlike other SNES cartridges which do use battery-backed SRAM instead of FLASH).
SNES Cart FLASH Backup

Baudrates
The BIOS seems to support max 2400 baud (old BIOS version) and 9600 baud (new BIOS version) - accordingly, there are probably also two different hardware versions of the SFC Modem.

Note
There's also another modem (which connects to cartridge slot):
SNES Cart X-Band (2400 baud Modem)

SNES Add-On SFC Modem - Data I/O

The modem is intended to be connected to controller port 2. RX Data, TX Data, and Modem Status are simultaneously transferred via three I/O lines. The overall transfer length (with ID bits) is 16-bit, however, after checking the ID bits, one can abbreviate the transfer to 9-bit length.

JOY2: (4017h.Bit0) - RX Data and ID Bits
```
  1st          RX Data Bit7    (0=High=Zero, 1=Low=One) ;\
  2nd          RX Data Bit6    (0=High=Zero, 1=Low=One) ;
  3rd          RX Data Bit5    (0=High=Zero, 1=Low=One) ; to be ignored when
  4th          RX Data Bit4    (0=High=Zero, 1=Low=One) ; no RX Data Present
  5th          RX Data Bit3    (0=High=Zero, 1=Low=One) ;
  6th          RX Data Bit2    (0=High=Zero, 1=Low=One) ;
  7th          RX Data Bit1    (0=High=Zero, 1=Low=One) ;
  8th          RX Data Bit0    (0=High=Zero, 1=Low=One) ;/
  9th          RX Data Present (0=High=None, 1=Low=Yes)
  10th         Unknown/Unused
  11th         Unknown/Unused
  12th         Unknown/Unused
  13th         ID Bit3 (always 0=High)
  14th         ID Bit2 (always 0=High)
  15th         ID Bit1 (always 1=Low)
  16th         ID Bit0 (always 1=Low)
  17th and up  Unknown/Unused (probably always whatever)
```
JOY4: (4017h.Bit1) - Modem Status
```
  1st          Unknown Flags Bit7  (1=Low=Busy or so, 0=Ready to get TX Data)
  2nd          Unknown Flags Bit6  (0=High=Error/Abort or so)
  3rd          Unknown Flags Bit5  (1=Low=Busy or so)
  4th          Unknown Flags Bit4  (1=Low=Busy or so)
  5th          Unknown Flags Bit3  Unused?
  6th          Unknown Flags Bit2  Unused?
  7th          Unknown Flags Bit1  Unused?
  8th          Unknown Flags Bit0  Unused?
  9th and up   Unknown/Unused (probably always whatever)
```
IOBIT (4201h.Bit7) - TX Data
1st bit should be output immediately after strobing 4016h.Output, 2nd..9th bit should be output immediately after reading 1st..8th data/status bits from 4017h.
```
  1st          TX Data Present (0=Low=Yes, 1=HighZ=None)
  2nd          TX Data Bit7    (0=Low=Zero, 1=HighZ=One) ;\
  3rd          TX Data Bit6    (0=Low=Zero, 1=HighZ=One) ; should be DATA
  4th          TX Data Bit5    (0=Low=Zero, 1=HighZ=One) ; when Data Present,
  5th          TX Data Bit4    (0=Low=Zero, 1=HighZ=One) ; or otherwise,
  6th          TX Data Bit3    (0=Low=Zero, 1=HighZ=One) ; should be FFh,
  7th          TX Data Bit2    (0=Low=Zero, 1=HighZ=One) ; or "R" or "C" ?
  8th          TX Data Bit1    (0=Low=Zero, 1=HighZ=One) ; (RTS/CTS or so?)
  9th          TX Data Bit0    (0=Low=Zero, 1=HighZ=One) ;/
  10th and up  Should be "1"   (1=HighZ)
```

SNES Add-On SFC Modem - Misc

"The Modem as far as I know only had one function and that was to allow you to do online betting via the official JRA (Japanese Horse Racing) online service. The modem ran on the NTT lines which probably means that NTT (Nippon Telecommunications) also had something to make out of this service or at least they thought they did :-D"
```
  JRA = Japan Racing Association (japanese horse racing)
  PAT = Personal Access Terminal (for telephone/online betting)
  NTT = Nippon Telegraph and Telephone (japanese telecommunications)
```

NTT JRA PAT (1997) (2400 Baud version) (J)
```
  baud rates seem to be 2400,1200 (see ROM 03:8910) (and "AT%B" strings)
  uses standard AT-commands (ATI0, ATS0?, ATD, ATX1, etc.)
  supports AMD FLASH only
  there are two ROM versions (SHVC-TJAJ-0 and SHVC-TJBJ-0)
```

NTT JRA PAT - Wide Baken Taiyou (1999) (9600 Baud version) (J)
```
  baud rates seem to be 9600,2400,1200 (see ROM 03:87F0) (and "AT%B" strings)
  uses standard AT-commands (ATI0, ATS0?, ATD, ATX1, etc.)
  supports AMD/ATMEL/SHARP FLASH
  there are two ROM versions (SHVC-TJDJ-0 and SHVC-TJEJ-0)
```

Zaitaku Touhyou System - SPAT4-Wide (1999 or so)
```
  unknown, reportedly also horse betting with NTT modem
  there is one ROM version (SHVC-TOBJ-0)
```

There is no special "modem-hardware" entry in cartridge header, but: note that all NTT/SFC modem BIOS have special "SHVC-Txxx-x" game codes.

SNES Add-On Voice-Kun (IR-transmitter/receiver for use with CD Players)

The Voice-Kun (sometimes called Voicer-Kun) from Koei is an Infrared transmitter/receiver. The transmitter part is used for controlling Audio CD Players (ie. to select and play tracks from Audio CDs that are included with supported games). The receiver part is used to "learn" IR-signals from different Remote Control manufacturers.

Controller Bits
The existing games expect the IR-unit connected to Port 2 (and a Joypad or Mouse in Port 1). The controller ID can be read via 4017h.Bit0 (serially, with STB/CLK signals). The actual IR-data is transferred via 4017h.Bit1/4201h.Bit7 (directly, without STB/CLK signals).
```
 4017h.Bit0:
  1st-12th    Unknown/unused (probably always 0=High?)
  13th-16th   ID Bits 3-0 (MSB First, 0=High=Zero) (always 0Dh)
  17th and up Unknown/unused (probably always whatever?)
```

```
4017h.Bit1:
 any bits    Infrared level (receiver) (0=High=Off, 1=Low=On)
4201h.Bit7:
 any bits    Infrared level (transmit) (0=Low=Off, 1=High=On)
```

Required Buttons

```
[  ]  Stop
[|>}  Play
[||]  Pause
[<<]  Previous Track
[>>]  Next Track
0..9  Numeric Digits
+10   Plus 10          ;\alternately either one of these
>10   Two-Digit-Input  ;/can be selected during configuration
```

The sequence for entering 2-digit Track numbers may vary greatly (+10, or >10 or -/-- buttons, to pressed before or after the low-digits), and, unknown if the Phillips "Toggle-Bit" is supported (needed for selecting track 11,22,33,etc. So far, one may expect problems with some CD players (though, as workaround, maybe the japanese GUI of the Voice-Kun games allows to select the starting track manually?).

Low Level Signals

```
Logical  ________--------------------________--------------------________
Physical ________||||||||||||||||||||________||||||||||||||||||||________
```

The Voice-Kun hardware is automatically modulating/demodulating the transmitted/received signals, so the SNES software does only need to deal with "Logical" signals.

Voice-Kun Games

```
Angelique Voice Fantasy (29 Mar 1996)
EMIT Vol. 1 - Toki no Maigo (25 Mar 1995)
EMIT Vol. 2 - Inochigake no Tabi (25 Mar 1995)
EMIT Vol. 3 - Watashi ni Sayonara wo (25 Mar 1995)
EMIT Value Set (EMIT Vol. 1-3) (15 Dec 1995)
```

Note - Soundware

Koei also made a number of "Soundware" games (mostly for other consoles/computers), which did also include Audio CDs or Audio Tapes.
For the SNES, Koei did release "Super Sangokushi II", originally on 15/09/1991, and re-released on 30 Mar 1995, at least one of that two releases (unclear which one) has been reportedly available with "Soundware" - unknown if that soundware version is Voice-Kun compatible, or (if it isn't compatible) unknown how else it was intended to be used...?)

SNES Cartridges

General Cartridge Info

SNES Cartridge ROM Header
SNES Cartridge PCBs
XXX SNES SRAM Mapping
SNES Cartridge ROM-Image Headers and File Extensions
SNES Cartridge ROM-Image Interleave
SNES Cartridge CIC Lockout Chip
SNES Cartridge Slot Pinouts

Basic Mapping Schemes

SNES Cart LoROM Mapping (ROM divided into 32K banks) (around 1500 games)
SNES Cart HiROM Mapping (ROM divided into 64K banks) (around 500 games)

Cartridges with External Programmable CPUs

SNES Cart SA-1 (programmable 65C816 CPU) (aka Super Accelerator) (35 games)
SNES Cart GSU-n (programmable RISC CPU) (aka Super FX/Mario Chip) (10 games)
SNES Cart Capcom CX4 (programmable RISC CPU) (Mega Man X 2-3) (2 games)

Cartridges with External Pre-Programmed CPUs

SNES Cart DSP-n/ST010/ST011 (pre-programmed NEC uPD77C25 CPU) (23 games)
SNES Cart Seta ST018 (pre-programmed ARM CPU) (1 game)

Cartridges with other custom chips

SNES Cart OBC1 (OBJ Controller) (1 game)
SNES Cart S-DD1 (Data Decompressor) (2 games)
SNES Cart SPC7110 (Data Decompressor) (3 games)
SNES Cart Unlicensed Variants

Cartridges with Real-Time Clocks

SNES Cart S-RTC (Realtime Clock) (1 game)
SNES Cart SPC7110 with RTC-4513 Real Time Clock (1 game)
Moreover, the Satellaview has a time function (allowing to receive the time via satellite dish). And, there are S-3520 (Seiko RTC's) in the Super Famicom Box and in the Nintendo Super System.

Special Non-game Cartridges/Expansions

SNES Cart Super Gameboy
SNES Cart Satellaview (satellite receiver & mini flashcard)
SNES Cart Data Pack Slots (satellaview-like mini-cartridge slot)
SNES Cart Nintendo Power (flashcard)
SNES Cart Sufami Turbo (Mini Cartridge Adaptor)
SNES Cart X-Band (2400 baud Modem)
SNES Cart FLASH Backup
SNES Cart Cheat Devices
SNES Cart Tri-Star (aka Super 8) (allows to play NES games on the SNES)
SNES Cart Pirate X-in-1 Multicarts (1)
SNES Cart Pirate X-in-1 Multicarts (2)
SNES Cart Copiers
SNES Cart CDROM Drive (not released)

SNES Cartridge ROM Header

The Cartridge header is mapped to 00FFxxh in SNES memory (near the exception vectors). In ROM-images it is found at offset 007Fxxh (LoROM), 00FFxxh (HiROM), or 40FFxxh (ExHiROM); add +200h to that offsets if "(imagesize AND 3FFh)=200h", ie. if there's an extra header from SWC/UFO/etc. copiers.

Cartridge Header (Area FFC0h..FFCFh)

```
FFC0h  Cartridge title (21 bytes, uppercase ascii, padded with spaces)
FFC0h  First byte of title (or 5Ch far-jump-opcode in Pirate X-in-1 Carts)
FFD4h  Last byte of title (or 00h indicating Early Extended Header)
FFD5h  Rom Makeup / ROM Speed and Map Mode (see below)
FFD6h  Chipset (ROM/RAM information on cart) (see below)
FFD7h  ROM size (1 SHL n) Kbytes (usually 8=256KByte .. 0Ch=4MByte)
        Values are rounded-up for carts with 10,12,20,24 Mbits
FFD8h  RAM size (1 SHL n) Kbytes (usually 1=2Kbyte .. 5=32Kbyte) (0=None)
FFD9h  Country (also implies PAL/NTSC) (see below)
FFDAh  Developer ID code  (00h=None/Homebrew, 01h=Nintendo, etc.) (33h=New)
FFDBh  ROM Version number (00h=First)
FFDCh  Checksum complement (same as below, XORed with FFFFh)
FFDEh  Checksum (all bytes in ROM added together; assume [FFDC-F]=FF,FF,0,0)
```

Extended Header (Area FFB0h..FFBFh) (newer carts only)

Early Extended Header (1993) (when [FFD4h]=00h; Last byte of Title=00h):

```
FFB0h  Reserved   (15 zero bytes)
```

Later Extended Header (1994) (when [FFDAh]=33h; Old Maker Code=33h):

```
FFB0h  Maker Code (2-letter ASCII, eg. "01"=Nintendo)
FFB2h  Game Code  (4-letter ASCII) (or old 2-letter padded with 20h,20h)
FFB6h  Reserved   (6 zero bytes)
FFBCh  Expansion FLASH Size (1 SHL n) Kbytes (used in JRA PAT)
FFBDh  Expansion RAM Size (1 SHL n) Kbytes (in GSUn games) (without battery?)
FFBEh  Special Version      (usually zero) (eg. promotional version)
```

Both Early and Later Extended Headers:

```
FFBFh  Chipset Sub-type (usually zero) (used when [FFD6h]=Fxh)
```

Note: The early-extension is used only with ST010/11 games.
If the first letter of the 4-letter Game Code is "Z", then the cartridge does have Satellaview-like Data Pack/FLASH cartridge slot (this applies ONLY to "Zxxx" 4-letter codes, not to old "Zx " space padded 2-letter codes).

Cartridge Header Variants

The BS-X Satellaview FLASH Card Files, and Sufami Turbo Mini-Cartridges are using similar looking (but not fully identical) headers (and usually the same .SMC file extension) than normal ROM cartridges. Detecting the content of .SMC files can be done by examining ID Strings (Sufami Turbo), or differently calculated checksum values (Satellaview). For details, see:
SNES Cart Satellaview (satellite receiver & mini flashcard)
SNES Cart Sufami Turbo (Mini Cartridge Adaptor)
Homebrew games (and copiers & cheat devices) are usually having several errors in the cartridge header (usually no checksum, zero-padded title, etc), they should (hopefully) contain valid entryoints in range 8000h..FFFEh. Many Copiers are using 8Kbyte ROM bank(s) - in that special case the exception vectors are located at offset 1Fxxh within the ROM-image.

CPU Exception Vectors (Area FFE0h..FFFFh)

```
FFE0h  Zerofilled (or ID "XBOO" for WRAM-Boot compatible files)
FFE4h  COP vector     (65C816 mode) (COP opcode)
FFE6h  BRK vector     (65C816 mode) (BRK opcode)
FFE8h  ABORT vector   (65C816 mode) (not used in SNES)
FFEAh  NMI vector     (65C816 mode) (SNES V-Blank Interrupt)
FFECh  ...
FFEEh  IRQ vector     (65C816 mode) (SNES H/V-Timer or External Interrupt)
FFF0h  ...
FFF4h  COP vector     (6502 mode)
FFF6h  ...
FFF8h  ABORT vector   (6502 mode) (not used in SNES)
FFFAh  NMI vector     (6502 mode)
FFFCh  RESET vector   (6502 mode) (CPU is always in 6502 mode on RESET)
FFFEh  IRQ/BRK vector (6502 mode)
```

Note: Exception Vectors are variable in SA-1 and CX4, and fixed in GSU.

Text Fields

The ASCII fields can use chr(20h..7Eh), actually they are JIS (with Yen instead backslash).

ROM Size / Checksum Notes

The ROM size is specified as "(1 SHL n) Kbytes", however, some cartridges contain "odd" sizes:

```
* Game uses 2-3 ROM chips (eg. one 8MBit plus one 2MBit chip)
* Game originally designed for 2 ROMs, but later manufactured as 1 ROM (?)
* Game uses a single 24MBit chip (23C2401)
```

In all three cases the ROM Size entry in [FFD7h] is rounded-up. In memory, the "bigger half" is mapped to address 0, followed by the "smaller half", then followed by mirror(s) of the smaller half. Eg. a 10MBit game would be rounded to 16MBit, and mapped (and checksummed) as "8Mbit + 4x2Mbit". In practice:

```
Title                       Hardware          Size   Checksum
Dai Kaiju Monogatari 2 (J)  ExHiROM+S-RTC     5MB    4MB + 4 x Last 1MB
Tales of Phantasia (J)      ExHiROM           6MB    <???>
Star Ocean (J)              LoROM+S-DD1       6MB    4MB + 2 x Last 2MB
Far East of Eden Zero (J)   HiROM+SPC7110+RTC 5MB    5MB
Momotaro Dentetsu Happy (J) HiROM+SPC7110     3MB    2 x 3MB
Sufami Turbo BIOS           LoROM in Minicart xx     without checksum
Sufami Turbo Games          LoROM in Minicart xx     without checksum
Dragon Ball Z - Hyper Dimension    LoROM+SA-1 3MB       Overdump 4MB
SD Gundam GNext (J)                LoROM+SA-1 1.5MB     Overdump 2MB
Megaman X2                         LoROM+CX4  1.5MB     Overdump 2MB
BS Super Mahjong Taikai (J)        BS           Overdump/Mirr+Empty
Demon's Crest... reportedly 12MBit ? but, that's bullshit ?

SPC7110 Title               ROM Size (Header value)    Checksum
Super Power League 4        2MB      (rounded to 2MB)  1x(All 2MB)
Momotaro Dentetsu Happy (J) 3MB      (rounded to 4MB)  2x(All 3MB)
Far East of Eden Zero (J)   5MB      (rounded to 8MB)  1x(All 5MB)
```

On-chip ROM contained in external CPUs (DSPn,ST01n,CX4) is NOT counted in the ROM size entry, and not included in the checksum.

Homebrew files often contain 0000h,0000h or FFFFh,0000h as checksum value.

ROM Speed and Map Mode (FFD5h)

```
Bit7-6 Always 0
Bit5   Always 1 (maybe meant to be MSB of bit4, for "2" and "3" MHz)
Bit4   Speed (0=Slow, 1=Fast)             (Slow 200ns, Fast 120ns)
Bit3-0 Map Mode
```

Map Mode can be:

```
0=LoROM/32K Banks             Mode 20 (LoROM)
1=HiROM/64K Banks             Mode 21 (HiROM)
2=LoROM/32K Banks + S-DD1     Mode 22 (mappable) "Super MMC"
3=LoROM/32K Banks + SA-1      Mode 23 (mappable) "Emulates Super MMC"
5=HiROM/64K Banks             Mode 25 (ExHiROM)
A=HiROM/64K Banks + SPC7110   Mode 25? (mappable)
```

Note: ExHiROM is used only by "Dai Kaiju Monogatari 2 (JP)" and "Tales of Phantasia (JP)".

Chipset (ROM/RAM information on cart) (FFD6h) (and some subclassed via FFBFh)

```
00h     ROM
01h     ROM+RAM
02h     ROM+RAM+Battery
x3h     ROM+Co-processor
x4h     ROM+Co-processor+RAM
x5h     ROM+Co-processor+RAM+Battery
x6h     ROM+Co-processor+Battery
x9h     ROM+Co-processor+RAM+Battery+RTC-4513
xAh     ROM+Co-processor+RAM+Battery+overclocked GSU1 ? (Stunt Race)
x2h     Same as x5h, used in "F1 Grand Prix Sample (J)" (?)
0xh     Co-processor is DSP    (DSP1,DSP1A,DSP1B,DSP2,DSP3,DSP4)
1xh     Co-processor is GSU    (MarioChip1,GSU1,GSU2,GSU2-SP1)
2xh     Co-processor is OBC1
3xh     Co-processor is SA-1
4xh     Co-processor is S-DD1
5xh     Co-processor is S-RTC
Exh     Co-processor is Other  (Super Gameboy/Satellaview)
Fxh.xxh Co-processor is Custom (subclassed via [FFBFh]=xxh)
Fxh.00h Co-processor is Custom (SPC7110)
Fxh.01h Co-processor is Custom (ST010/ST011)
Fxh.02h Co-processor is Custom (ST018)
Fxh.10h Co-processor is Custom (CX4)
```

In practice, following values are used:

```
00h     ROM             ;if gamecode="042J" --> ROM+SGB2
01h     ROM+RAM (if any such produced?)
02h     ROM+RAM+Battery ;if gamecode="XBND" --> ROM+RAM+Batt+XBandModem
                        ;if gamecode="MENU" --> ROM+RAM+Batt+Nintendo Power
03h     ROM+DSP
04h     ROM+DSP+RAM (no such produced)
05h     ROM+DSP+RAM+Battery
13h     ROM+MarioChip1/ExpansionRAM (and "hacked version of OBC1")
14h     ROM+GSU+RAM                   ;\ROM size up to 1MByte -> GSU1
15h     ROM+GSU+RAM+Battery           ;/ROM size above 1MByte -> GSU2
1Ah     ROM+GSU1+RAM+Battery+Fast Mode? (Stunt Race)
25h     ROM+OBC1+RAM+Battery
32h     ROM+SA1+RAM+Battery (?) "F1 Grand Prix Sample (J)"
34h     ROM+SA1+RAM (?) "Dragon Ball Z - Hyper Dimension"
35h     ROM+SA1+RAM+Battery
43h     ROM+S-DD1
45h     ROM+S-DD1+RAM+Battery
55h     ROM+S-RTC+RAM+Battery
E3h     ROM+Super Gameboy      (SGB)
E5h     ROM+Satellaview BIOS   (BS-X)
F5h.00h ROM+Custom+RAM+Battery     (SPC7110)
F9h.00h ROM+Custom+RAM+Battery+RTC (SPC7110+RTC)
F6h.01h ROM+Custom+Battery         (ST010/ST011)
F5h.02h ROM+Custom+RAM+Battery     (ST018)
F3h.10h ROM+Custom                 (CX4)
```

Country (also implies PAL/NTSC) (FFD9h)

```
00h -  International (eg. SGB)  (any)
00h J  Japan                    (NTSC)
01h E  USA and Canada           (NTSC)
02h P  Europe, Oceania, Asia    (PAL)
03h W  Sweden/Scandinavia       (PAL)
04h -  Finland                  (PAL)
05h -  Denmark                  (PAL)
06h F  France                   (SECAM, PAL-like 50Hz)
07h H  Holland                  (PAL)
08h S  Spain                    (PAL)
09h D  Germany, Austria, Switz  (PAL)
0Ah I  Italy                    (PAL)
0Bh C  China, Hong Kong         (PAL)
0Ch -  Indonesia                (PAL)
0Dh K  South Korea              (NTSC) (North Korea would be PAL)
0Eh A  Common (?)               (?)
0Fh N  Canada                   (NTSC)
10h B  Brazil                   (PAL-M, NTSC-like 60Hz)
11h U  Australia                (PAL)
12h X  Other variation          (?)
13h Y  Other variation          (?)
14h Z  Other variation          (?)
```

Above shows the [FFD9h] value, and the last letter of 4-character game codes.

Game Codes (FFB2h, exists only when [FFDAh]=33h)

```
"xxxx"  Normal 4-letter code (usually "Axxx") (or "Bxxx" for newer codes)
"xx  "  Old 2-letter code (space padded)
"042J"  Super Gameboy 2
"MENU"  Nintendo Power FLASH Cartridge Menu
"Txxx"  NTT JRA-PAT and SPAT4 (SFC Modem BIOSes)
"XBND"  X-Band Modem BIOS
"Zxxx"  Special Cartridge with satellaview-like Data Pack Slot
```

The last letter indicates the region (see Country/FFD9h description) (except in 2-letter codes and "MENU"/"XBND" codes).

SNES Cartridge PCBs

Cartridge PCB Naming (eg. SHVC-XXXX-NN)

Prefix

```
SHVC  Normal cartridge (japan, usa, europe)
SNSP  Special PAL version (for SA1 and S-DD1 with built-in CIC)
BSC   BIOS (or game cartridge) with external Satellaview FLASH cartridge slot
MAXI  Majesco Sales Inc cartridge (Assembled in Mexico)
MJSC  Majesco Sales Inc cartridge (Assembled in Mexico)
WEI   Whatever? (Assembled in Mexico)
EA    Electronics Arts cartridge
```

First Character

```
1   One ROM chip (usually 36pin, sometimes 32pin)
Y   Two 4Mbit ROM chips     (controlled by 74LS00)
2   Two 8Mbit ROM chips     (controlled by 74LS00,MAD-1,etc.)
B   Two 16Mbit ROM chips    (controlled by 74LS00 or MAD-1)
L   Two 32Mbit ROM chips    (controlled by SPC7110F,S-DD1 or MAD-1)
3   Three 8Mbit ROM chips   (controlled by 74LS139) (decoder/demultiplexer)
4   Four ROM chips (used only for 4PVnn/4QW EPROM prototype boards)
8   Eight ROM chips (used only for 8PVnn/8Xnn EPROM prototype boards)
```

Second Character(s)

```
A   LoRom (A15 / Pin40 not connected to ROM)   (uh, 1A3B-20 ?)
B   LoRom plus DSP-N chip
C   LoRom plus Mario Chip 1               62pin  (and 36pin ROM) (no X1)
CA  LoRom plus GSU-1                      62pin  (and 32pin ROM)
CB  LoRom plus GSU-2 or GSU-2-SP1         62pin  (and 40pin ROM)
DC  LoRom plus CX4                        62pin  (and 32pin ROMs)
DH  HiRom plus SPC7110F                   62pin  (and 32pin+44pin ROMs)
DE  LoRom plus ST018                      62pin  (and 32..40pin ROM possible)
DS  LoRom plus ST010/ST011                62pin  (and 32..36pin ROM possible)
E   LoRom plus OBC1 chip                  62pin  (and 32pin ROMs)
J   HiRom (A15 / Pin40 is connected to ROM)
K   HiRom plus DSP-N chip
L   LoRom plus SA1 chip       62pin  (and 44pin ROM) (16bit data?)
N   LoRom plus S-DD1 chip     62pin  (and 44pin ROM)
P   LoRom with 2 prototype EPROMs (=unlike ROM A16..Ahi,/CS)
PV  WhateverRom with 4 prototype EPROMs (=unlike ROM A16..Ahi,/CS)
Q   WhateverRom prototype (see book2.pdf)
QW  WhateverRom prototype (see book2.pdf)
RA  LoRom plus GSU1A with prototype EPROMs (=unlike ROM A16..Ahi,/CS)  62pin
X   WhateverRom prototype (see book2.pdf)
```

Third Character

```
0   No SRAM
1   2Kx8 SRAM (usually narrow 24pin DIP, sometimes wide 24pin DIP)
2   prototype variable size SRAM
3   8Kx8 SRAM (usually wide 28pin DIP)
5   32Kx8 SRAM (usually wide 28pin DIP)
6   64Kx8 SRAM (32pin SMD, found on boards with GSU)
8   64Kx8 SRAM (in one SA1 cart) (seems to be a 64Kx8 chip, not 256Kx8)
```

Forth Character

```
N   No battery
B   Battery (with Transistor+Diodes or MM1026/MM1134 chip)
M   Battery (with MAD-1 chip; or with rare MAD-R chip)
X   Battery (with MAD-2 chip; maybe amplifies X1 oscillator for DSP1B chips)
C   Battery and RTC-4513
R   Battery and S-RTC
F   FLASH Memory (instead of SRAM) (used by JRA-PAT and SPAT4)
```

Fifth Characters (only if cart contains RAM that is NOT battery-backed)

```
5S  32Kx8 SRAM (for use by GSU, not battery backed)
6S  64Kx8 SRAM (for use by GSU, not battery backed)
7S  64Kx8 or 128Kx8 SRAM (for use by GSU, not battery backed)
9P  512Kx8 PSRAM (32pin 658512LFP-85) (for satellaview)
(the black-blob Star Fox PCB also contains RAM, but lacks the ending "nS")
```

Suffix

```
-NN revision number (unknown if this indicates any relevant changes)
```

Other Cartridge PCBs (that don't follow the above naming system)

```
CPU2 SGB-R-10  Super Gameboy (1994)
SHVC-MMS-02    Nintendo Power FLASH Cartridge (1997)
SHVC-MMSA-1    Nintendo Power FLASH Cartridge (19xx) ???
SHVC-SGB2-01   Super Gameboy 2 (1998)
SHVC-1C0N      Star Fox (black blob version) (PCB name lacks ending nS-NN)
SHVC TURBO     Sufami Turbo BASE CASSETTE (Bandai)
<unknown?>     Sufami Turbo game cartridges
123-0002-16    X-Band Modem (1995 by Catapult / licensed by Nintendo)
BSMC-AF-01     Satellaview Mini FLASH Cartridge (plugged into BIOS cartridge)
BSMC-CR-01     Satellaview Mini FLASH Cartridge (???)
GPC-RAMC-4M    SRAM Cartridge (without ROM)?
GPC-RAMC-S1    SRAM Cartridge (without ROM)?
GS 0871-102    Super Famicom Box multi-game cartridge
NSS-01-ROM-A   Nintendo Super System (NSS) cartridge
NSS-01-ROM-B   Nintendo Super System (NSS) cartridge
NSS-01-ROM-C   Nintendo Super System (NSS) cartridge
NSS-X1-ROM-C   Rebadged NSS-01-ROM-C board (plus battery/sram installed)
```

ROM Chips used in SNES cartridges

```
2Mbit 256Kbyte        LH532 TC532 N-2001 (2nd chip/2A0N) (+SGB) (+Sufami)
4Mbit 512Kbyte 23C401 LH534 TC534 HN623n4 HN623x5 23C4001 LH5S4 CAT534 CXK384
8Mbit  1Mbyte  23C801 LH538 TC538 HN623n8 23C8001 TC23C8003 CAT548
16Mbit 2Mbyte  23C1601 LH537 LHMN7 TC5316 M5316
24Mbit 3Mbyte  23C2401 (seen on SHVC-1J3M board)
32Mbit 4Mbyte  23C3201 LH535 LHMN5 M5332 23C3202/40pin/SA1 N-32000/44pin/DD1
```

DIP vs SMD vs Blobs

Most SNES carts are using DIP chips. SMD chips are used only in carts with coprocessors (except S-RTC, DSP-n, ST01n). Black blobs are found in several pirate

carts (and in Star Fox, which contains Nintendo's Mario Chip 1, so it's apparently not a pirate cart).

SNES Cartridge ROM-Image Headers and File Extensions

Below file headers are dated back to back-up units, which allowed to load ROM-images from 1.44MB floppy disks into RAM, larger images have been split into "multi files".
Many of these files do have 512-byte headers. The headers don't contain any useful information. So, if they are present: Just ignore them. Best way to detect them is: "IF (filesize AND 3FFh)=200h THEN HeaderPresent=True" (Headerless cartridges are always sized N*1024 bytes, Carts with header are N*1024+512 bytes).

.SMC - Super MagiCom (by Front Far East)
This extension is often used for ANY type of SNES ROM-images, including for SWC files.

.SWC - Super Wild Card (SWC) Header (by Front Far East)
```
000h-001h  ROM Size (in 8Kbyte units)
002h       Program execution mode
            Bit   Expl.
            7     Entrypoint (0=Normal/Reset Vector, 1=JMP 8000h)
            6     Multi File (0=Normal/Last file, 1=Further file(s) follow)
            5     SRAM mapping    (0=mode20, 1=mode21)
            4     Program mapping (0=mode20, 1=mode21)
            3-2   SRAM Size  (0=32Kbytes, 1=8Kbytes, 2=2Kbytes, 3=None)
            1     Reserved   (zero)
            0     Unknown    (seems to be randomly set to 0 or 1)
003h       Reserved (zero) (but, set to 01h in homebrew "Pacman" and "Nuke")
004h-007h  Reserved (zero)
008h-009h  SWC File ID (AAh, BBh)
00Ah       File Type (04h=Program ROM, 05h=Battery SRAM, 08h=real-time save)
00Bh-1FFh  Reserved (zero)
```

.FIG - Pro Fighter (FIG) header format (by China Coach Ltd)
```
000h-001h  ROM Size (in 8Kbyte units)
002h       Multi File (00h=Normal/Last file, 40h=Further file(s) follow)
                      (02h=Whatever, used in homebrew Miracle,Eagle,Cen-Dem)
003h       ROM Mode   (00h=LoROM, 80h=HiROM)
004h-005h  DSP1/SRAM Mode (8377h=ROM, 8347h=ROM+DSP1, 82FDh=ROM+DSP1+SRAM)
006h-1FFh  Reserved (zero) (or garbage at 01FCh in homebrew Darkness Demo)
```

.BIN - Raw Binary
Contains a raw ROM-image without separate file header.

.078 - Game Doctor file name format (by Bung)
Contains a raw ROM-image without separate file header.
Information about multi files is encoded in the "SFxxyyyz.078" filenames.
```
SF   Abbreviation for Super Famicom
xx   Image size in Mbit (2,4,8,16,32) (1-2 chars, WITHOUT leading zero)
yyy  Game catalogue number (or random number if unknown)
z    Indicates multi file (A=first, B=second, etc.)
078  File extension (should be usually 078)
```

.MGD - Multi Game Doctor ? (by Bung)
Format Unknown?

Another Game Doctor version...
```
000h-00Fh  ID "GAME DOCTOR SF 3"
010h       Unknown (80h)      ;-SRAM size limit
011h       Unknown (20h)      ;\
012h       Unknown (21h)      ; DRAM mapping related
013h-018h  Unknown (6x60h)    ;
019h       Unknown (20h)      ;
01Ah       Unknown (21h)      ;
01Bh-028h  Unknown (14x60h)   ;/
029h-02Ah  Zero               ;-SRAM mapping related
 011h-020h  512Kbyte DRAM chunk, mapped to upper 32Kbyte of Bank 0xh-Fxh
 021h-024h  512Kbyte DRAM chunk, mapped to lower 32Kbyte of Bank 4xh-7xh
 025h-028h  512Kbyte DRAM chunk, mapped to lower 32Kbyte of Bank Cxh-Fxh
 029h-02Ah  SRAM Flags (bit0-15 = Enable SRAM at 6000-7000 in banks 0xh-Fxh)
02Bh-1FFh  Zero (Reserved)
```

Superufo
```
000h       Unknown (20h or 40h)   ;maybe ROM size in 8K units   ?
001h-007h  Zero
008h-00Fh  ID "SUPERUFO"
010h       Unknown (01h)
011h       Unknown (02h or 04h)   ;maybe rom speed              ?
012h       Unknown (E1h or F1h)   ;MSB=chipset (Exh or Fxh)     ?
013h       Unknown (00h)
014h       Unknown (01h)
015h       Unknown (03h)
016h       Unknown (00h)
017h       Unknown (03h)
018h-1FFh  Zero
```

.SFC - Nintendo Developer File (Nintendo)
Contains a raw ROM-image without separate file header.
Information about multi files is encoded in the "NnnnVv-N.SFC" filenames.
```
Nnnn  Game code (4 letters)
Vv    ROM Version
N     Disk Number (0=First)
SFC   Fixed extension (Super FamiCom)
```
This is how Nintendo wanted developers to name their files.

NSRT Header (can be generated by Nach's NSRT tool)
This format stores some additional information in a formerly unused 32-byte area near the end of the 512-byte copier headers.

```
  1D0h  Unknown/unspecified (LSB=01h..03h, MSB=00h..0Dh) ;maybe ROM mapping
  1D1h  Unknown/unspecified   ;maybe title and/or NSRT version
  1E8h  ID1 "NSRT"
  1ECh  ID2 16h (22 decimal)
  1EDh  Controllers (MSB=Port1, LSB=Port2)
  1EEh  Checksum (sum of bytes at [1D0h..1EDh]+FFh)
  1EFh  Checksum Complement (Checksum XOR FFh)
```

Controller Values:

```
  00h Gamepad
  01h Mouse
  02h Mouse or Gamepad
  03h Super Scope
  04h Super Scope or Gamepad
  05h Justifier
  06h Multitap
  07h Mouse, Super Scope, or Gamepad
  08h Mouse or Multitap
  09h Lasabirdie
  0Ah Barcode Battler
  0Bh..0Fh Reserved
```

Most copiers also include a parallel PC port interface, allowing your PC to control the unit and store images on your hard drive.
Copier's contain DRAM from 1 Megabyte to 16 Megabytes, 8MegaBits to 128MegaBits respectively. This is the reason why they are so expensive.

SNES Cartridge ROM-Image Interleave

Some ROM images are "interleaved", meaning that their content is ordered differently as how it appears in SNES memory. The interleaved format dates back to old copiers, most modern tools use/prefer normal ROM-images without interleave, but old interleaved files may still show up here and there.

Interleave used by Game Doctor & UFO Copiers
These copiers use interleave so that the ROM Header is always stored at file offset 007Fxxh. For HiROM files, interleave is applied as so: store upper 32K of all 64K banks, followed by lower 32K of all 64K banks (which moves the header from 00FFxxh to 007Fxxh). For example, with a 320Kbyte ROM, the ten 32K banks would be ordered as so:

```
  0,1,2,3,4,5,6,7,8,9 - Original
  1,3,5,7,9,0,2,4,6,8 - Interleaved
```

For LoROM files, there's no interleave applied (since the header is already at 007Fxxh).
Detecting an interleaved file could be done as so:

```
  Header must be located at file offset 007Fxxh (ie. in LoROM fashion)
  Header must not be a Sufami Turbo header (=title "ADD-ON BASE CASSETE")
  Header must not be a Satellaview header (=different chksum algorithm)
  Header should not contain corrupted entries
  The "Map Mode" byte at "[007FD5h] ANDed with 0Fh" is 01h,05h,0Ah (=HiROM)
```

If so, the file is interleaved (or, possibly, it's having a corrupted header with wrong map mode setting).

Interleave used by Human Stupidity
There are interleaving & deinterleaving tools, intended to convert normal ROM-images to/from the format used by above copiers. Using that tools on files that are already in the desired format will result in messed-up garbage. For example, interleaving a 320Kbyte file that was already interleaved:

```
  1,3,5,7,9,0,2,4,6,8 - Interleaved
  3,7,0,4,8,1,5,9,2,6 - Double-Interleaved
```

Or, trying to deinterleave a 320Kbyte file that wasn't interleaved:

```
  0,1,2,3,4,5,6,7,8,9 - Original
  5,0,6,1,7,2,8,3,9,4 - Mis-de-interleaved
```

One can eventually repair such files by doing the opposite (de-)interleaving action. Or, in worst case, the user may repeat the wrong action, ending up with a Triple-Interleaved, or Double-mis-de-interleaved file.
Another case of stupidity would be applying interleave to a LoROM file (which would move the header <away from> 007Fxxh towards the middle of the file, ie. the opposite of the intended interleaving effect of moving it <to> 007Fxxh).

ExHiROM Files
ExHiROM Files are also having the data ordered differently as in SNES memory. However, in this special case, the data SHOULD be ordered as so. The ordering is: Fast HiROM (4Mbytes, banks C0h..FFh), followed by Slow HiROM banks (usually 1-2MByte, banks 40h..4Fh/5Fh) (of which, the header and exception vectors are in upper 32K of the first Slow HiROM bank, ie. at file offset 40FFxxh). There are only 2 games using the ExHiROM format:

```
  Dai Kaiju Monogatari 2 (JP) (5Mbytes) PCB: SHVC-LJ3R-01
  Tales of Phantasia (JP) (6Mbytes)     PCB: SHVC-LJ3M-01
```

The ExHiROM ordering is somewhat "official" as it was defined in Nintendo's developer manuals. Concerning software, the ordering does match-up with the checksum calculation algorithm (8MB chksum across 4MB plus mirror(s) of remaining 1-2MB). Concerning hardware, the ordering may have been 'required' in case Nintendo did (or planned to) use "odd" sized 5Mbyte/6Mbyte-chips (they DID produce cartridges with 3MByte/24Mbit chips).

SNES Cartridge CIC Lockout Chip

SNES cartridges are required to contain a CIC chip (security chip aka lockout chip). The CIC is a small 4bit CPU with built-in ROM. An identical CIC is located in the SNES console. The same 4bit CPU (but with slightly different code in ROM) is also used in NES consoles/cartridges.
The CIC in the console is acting as "lock", and that in the cartridge is acting as "key". The two chips are sending random-like bitstreams to each other, if the data (or transmission timing) doesn't match the expected values, then the "lock" issues a RESET signal to the console. Thereby rejecting cartridges without CIC chip (or such with CICs for wrong regions).

CIC Details
SNES Cartridge CIC Pseudo Code
SNES Cartridge CIC Instruction Set
SNES Cartridge CIC Notes
SNES Cartridge CIC Versions
SNES Pinouts CIC Chips

CIC Disable
SNES Common Mods

SNES Cartridge CIC Pseudo Code

CicMain

```
  CicInitFirst, CicInitTiming, CicRandomSeed, CicInitStreams
  time=data_start, a=1, noswap=1, if snes then noswap=0
 mainloop:
  for x=a to 0Fh
    if nes then Wait(time-5), else if snes then (time-7)     ;\verify idle
    if (nes_6113=0) and (P0.0=1 or P0.1=1) then Shutdown     ;/
    Wait(time+0)                                             ;\
    if (console xor snes) then a=[00h+x].0, else a=[10h+x].0 ; output data
    if noswap then P0.0=a, else P0.1=a                       ;/
    Wait(time+2-data_rx_error)                               ;\
    if (console xor snes) then a=[10h+x].0, else a=[00h+x].0 ; verify input
    if noswap then a=(a xor P0.1), else a=(a xor P0.0)       ;
    if a=1 then Shutdown                                     ;/
    Wait(time+3)                                             ;\output idle
    if noswap then P0.0=0, else P0.1=0                       ;/
    if snes then time=time+92, else if nes then time=time+79
  next x
  CicMangle(00h), CicMangle(10h)                        ;\mangle
  if snes then CicMangle(00h), CicMangle(10h)           ; (thrice on SNES)
  if snes then CicMangle(00h), CicMangle(10h)           ;/
  if snes then noswap=[17h].0   ;eventually swap input/output pins (SNES only)
  a=[17h]
  if a=0 then a=1, time=time+2
  if snes then time=time+44, else if nes then time=time+29
  goto mainloop
```

CicMangle(buf)

```
  for i=[buf+0Fh]+1 downto 1
    a=[buf+2]+[buf+3h]+1
    if a<10h then x=[buf+3], [buf+3]=a, a=x, x=1, else x=0
    [buf+3+x]=[buf+3+x]+a
    for a=x+6 to 0Fh, [buf+a]=[buf+a]+[buf+a-1]+1, next a
    a=[buf+4+x]+8, if a<10h then [buf+5+x]=[buf+5+x]+a, else [buf+5+x]=a
    [buf+4+x]=[buf+4+x]+[buf+3+x]
    [buf+1]=[buf+1]+i
    [buf+2]=NOT([buf+2]+[buf+1]+1)
    time=time+84-(x*6)
  next i
```

Note: All values in [buf] are 4bit wide (aka ANDed with 0Fh).

CicInitFirst

```
  timer=0                      ;reset timer (since reset released)
  P0=00h
  console=P0.3                 ;get console/cartridge flag
  if console
    while P0.2=1, r=r+1        ;get 4bit random seed (capacitor charge time)
    P1.1=1, P1.1=0             ;issue reset to CIC in cartridge
    timer=0                    ;reset timer (since reset released)
  if nes_6113 and (console=1)
    Wait(3), nes_6113_in_console=1, P0.0=1      ;request special 6113 mode
  if nes_6113 and (console=0)
    Wait(6), nes_6113_in_console=P0.1           ;check if 6113 mode requested
```

CicRandomSeed

```
  time=seed_start
  for i=0 to 3                 ;send/receive 4bit random seed (r)
    bit=((i+3) and 3)          ;bit order is 3,0,1,2 (!)
    if console=1 Wait(time+0+i*15), P0.0=r.bit, Wait(time+3+i*15), P0.0=0 ;send
    if console=0 Wait(time+2+i*15), r.bit=P0.1                            ;recv
  next i
```

CicInitStreams

```
  if snes
    if ntsc then x=9, else if pal then x=6
    [01h..0Fh]=B,1,4,F,4,B,5,7,F,D,6,1,E,9,8   ;init stream from cartridge (!)
    [11h..1Fh]=r,x,A,1,8,5,F,1,1,E,1,0,D,E,C   ;init stream from console   (!)
  if nes_usa                ;3193A
    [01h..0Fh]=1,9,5,2,F,8,2,7,1,9,8,1,1,1,5   ;init stream from console
    [11h..1Fh]=r,9,5,2,1,2,1,7,1,9,8,5,7,1,5   ;init stream from cartridge
    if nes_6113_in_console then overwrite [01h]=5 or so ???   ;special-case
  if nes_europe             ;3195A
    [01h..0Fh]=F,7,B,E,F,8,2,7,D,7,8,E,E,1,5   ;init stream from console
    [11h..1Fh]=r,7,B,D,1,2,1,7,E,6,7,A,7,1,5   ;init stream from cartridge
  if nes_hongkong_asia      ;3196A
    [01h..0Fh]=E,6,A,D,F,8,2,7,E,6,7,E,E,E,A   ;init stream from console
    [11h..1Fh]=r,6,A,D,E,D,E,8,E,6,7,A,7,1,5   ;init stream from cartridge
  if nes_uk_italy_australia ;3197A
    [01h..0Fh]=3,5,8,9,3,7,2,8,8,6,8,5,E,E,B   ;init stream from console
    [11h..1Fh]=r,7,9,A,A,1,6,8,5,8,9,1,5,1,7   ;init stream from cartridge
  if nes_famicombox         ;3198A
    (unknown)
```

Note: In most cases, the PAL region changes are simply inverted or negated NTSC values (not/neg), except, one NES-EUR value, and most of the NES-UK values are somehow different. The rev-engineered NES-UK values may not match the exact original NES-UK values (but they should be working anyways).

CicInitTiming

```
  if snes_d411           -> seed_start=630, data_start=817   ;snes/ntsc
  if snes_d413           -> (unknown?) (same as d411?)       ;snes/pal
  if nes_3193            -> (seems to be same as nes_3195?)  ;nes/usa (v1)
  if nes_3195            -> seed_start=32, data_start=200    ;nes/europe
  if nes_3196            -> (unknown?)                       ;nes/asia
  if nes_3197            -> (unknown?) ("burns five")        ;nes/uk
  if nes_6113            -> seed_start=32, data_start=201    ;nes/usa (v2)
  if nes_6113_in_console -> seed_start=33, data_start=216    ;nes/special
  if nes_tengen          -> seed_start=32, data_start=201    ;nes/cic-clone
  ;now timing errors...
  data_rx_error=0  ;default
```

```
  if console=0 and nes_3193a -> randomly add 0 or 0.25 to seed_start/data_start
  if console=0 and snes_d413 -> always add 1.33 to seed_start/data_start (bug)
  if console=0 and nes_6113  -> data_rx_error=1 (and maybe +1.25 on seed/data?)
  if other_chips & chip_revisions -> (unknown?)
```

Note: 3197 reportedly "burns five extra cycles before initialization", but unknown if that is relative to 3193 <or> 3195 timings, and unknown if it applies to <both> seed_start and data_start, and unknown if it means 1MHz <or> 4MHz cycles.
Note: The "data_rx_error" looks totally wrong, but it is somewhat done intentionally, so there might be a purpose (maybe some rounding, in case 6113 and 3193 are off-sync by a half clock cycle, or maybe an improper bugfix in case they are off-sync by 1 or more cycles).

Wait(time)
Wait until "timer=time", whereas "timer" runs at 1MHz (NES) or 1.024MHz (SNES). The "time" values are showing the <completion> of the I/O opcodes (ie. the I/O opcodes <begin> at "time-1").

Shutdown (should never happen, unless cartridge is missing or wrong region)

```
  a=0, if nes then time=830142, else if snes then time=1037682
 endless_loop:          ;timings here aren't 100.000% accurate
  if nes_3195 then time=xlat[P1/4]*174785  ;whereas, xlat[0..3]=(3,2,4,5)
  if (console=0) and (snes or nes_6113) then P0=03h, P1=01h
  if (console=1) then P1=a, Wait(timer+time), a=a xor 4  ;toggle reset on/off
  goto endless_loop
```

SNES Cartridge CIC Instruction Set

CIC Registers

```
  A  4bit Accumulator
  X  4bit General Purpose Register
  L  4bit Pointer Register (lower 4bit of 6bit HL)
  H  2bit Pointer Register (upper 2bit of 6bit HL)
  C  1bit Carry Flag (changed ONLY by "set/clr c", not by "add/adc" or so)
  PC 10bit Program Counter (3bit bank, plus 7bit polynomial counter)
```

CIC Memory

```
  ROM   512x8bit (program ROM) (NES/EUR=768x8) (max 1024x8 addressable)
  RAM   32x4bit  (data RAM) (max 64x4 addressable)
  STACK 4x10bit  (stack for call/ret opcodes)
  PORTS 4x4bit   (external I/O ports & internal RAM-like ports) (max 16x4)
```

Newer CIC Opcodes (6113, D411) (and probably F411,D413,F413)

```
  00      nop             no operation (aka "addsk A,0" opcode)
  00+n    addsk  A,n      add, A=A+n, skip if result>0Fh
  10+n    cmpsk  A,n      compare, skip if A=n
  20+n    mov    L,n      set L=n
  30+n    mov    A,n      set A=n
  40      mov    A,[HL]   set A=RAM[HL]
  41      xchg   A,[HL]   exchange A <--> RAM[HL]
  42      xchgsk A,[HL+]  exchange A <--> RAM[HL], L=L+1, skip if result>0Fh
  43      xchgsk A,[HL-]  exchange A <--> RAM[HL], L=L-1, skip if result<00h
  44      neg    A        negate, A=0-A                ;(used by 6113 mode)
  45      ?
  46      out    [L],A    output, PORT[L]=A
  47      out    [L],0    output, PORT[L]=0
  48      set    C        set carry, C=1
  49      clr    C        reset carry, C=0
  4A      mov    [HL],A   set RAM[HL]=A
  4B      ?
  4C      ret             return, pop PC from stack
  4D      retsk           return, pop PC from stack, skip
  4E+n    ?
  52      movsk  A,[HL+]  set A=RAM[HL], L=L+1, skip if result>0Fh
  53      ?                    (guess: movsk  A,[HL-])
  54      not    A        complement, A=A XOR 0Fh
  55      in     A,[L]    input, A=PORT[L]
  56      ?
  57      xchg   A,L      exchange A <--> L
  58+n    ?
  5C      mov    X,A      set X=A
  5D      xchg   X,A      exchange X <--> A
  5E      ???             "SPECIAL MYSTERY INSTRUCTION" ;(used by 6113 mode)
  5F      ?
  60+n    testsk [HL].n   skip if RAM[HL].Bit(n)=1
  64+n    testsk A.n      skip if A.Bit(n)=1
  68+n    clr    [HL].n   set RAM[HL].Bit(n)=0
  6C+n    set    [HL].n   set RAM[HL].Bit(n)=1
  70      add    A,[HL]   add, A=A+RAM[HL]
  71      ?                    (guess: addsk  A,[HL])
  72      adc    A,[HL]   add with carry, A=A+RAM[HL]+C
  73      adcsk  A,[HL]   add with carry, A=A+RAM[HL]+C, skip if result>0Fh
  74+n    mov    H,n      set H=n  ;2bit range, n=0..3 only (used: 0..1 only)
  78+n mm jmp    nmm      long jump, PC=nmm
  7C+n mm call   nmm      long call, push PC+2, PC=nmm
  80+nn   jmp    nn       short jump, PC=(PC AND 380h)+nn
  -       reset           PC=000h
```

Note: "skip" means "do not execute next instruction"

Older CIC Opcodes (3195) (and probably 3193,3196,3197,etc.)

```
  Exchanged opcodes 48 <--> 49 (set/clr C)
  Exchanged opcodes 44 <--> 54 (neg/not A)
  ROM Size is 768x8 (although only 512x8 are actually used)
```

Note
The CIC is a 4bit Sharp CPU (maybe a Sharp SM4, but no datasheet exists) (the instruction seems to be an older version of that in the Sharp SM5K1..SM5K7 datasheets).

SNES Cartridge CIC Notes

Program Counter (PC)
The 10bit PC register consists of a 3bit bank (which gets changed only by call/jmp/ret opcodes), and a 7bit polynomial counter (ie. not a linear counter). After fetching opcode bytes, PC is "incremented" as so:

```
PC = (PC AND 380h) + (PC.Bit0 XOR PC.Bit1)*40h + (PC AND 7Eh)/2
```

Ie. the lower 7bit will "increment" through 127 different values (and wrap to 00h thereafter). Address 7Fh is unused (unless one issues a JMP 7Fh opcode, which would cause the CPU to hang on that address).

```
Format     <------------- Valid Address Area ---------->      <--Stuck-->
Linear     00 01 02 03 04 05 06 07 08 09 0A ... 7C 7D 7E  or  7F 7F 7F 7F
Polynomial 00 40 60 70 78 7C 7E 3F 5F 6F 77 ... 05 02 01  or  7F 7F 7F 7F
```

To simplify things, programming tools like assemblers/disassemblers may use "normal" linear addresses (and translate linear/polynomial addressses when needed - the polynomial addresses are relevant only for encoding bits in jmp/call opcodes, and for how the data is physically arranged in the chip ROMs and in ROM-images).

ROM-Images
The existing ROM-images are .txt files, containing "0" and "1" BITS in ASCII format, arranged as a 64x64 (or 96x64) matrix (as seen in decapped chips).

```
Line 1..32   --->   Address X+9Fh..80h            ;\Lines (Y)
Line 33..64  --->   Address X+1Fh..00h            ;/
Column  1+(n*W) --> Data Bit(n) of Address 000h+Y ;\  ;\
Column  2+(n*W) --> Data Bit(n) of Address 020h+Y ;   ; Columns (X)
Column  3+(n*W) --> Data Bit(n) of Address 040h+Y ;   ;
Column  4+(n*W) --> Data Bit(n) of Address 060h+Y ;   ; chips with 200h-byte
Column  5+(n*W) --> Data Bit(n) of Address 100h+Y ;   ; (W=8) (64x64 bits)
Column  6+(n*W) --> Data Bit(n) of Address 120h+Y ;   ;
Column  7+(n*W) --> Data Bit(n) of Address 140h+Y ;   ;
Column  8+(n*W) --> Data Bit(n) of Address 160h+Y ;   ;/
Column  9+(n*W) --> Data Bit(n) of Address 200h+Y ;
Column 10+(n*W) --> Data Bit(n) of Address 220h+Y ;  chips with 300h-byte
Column 11+(n*W) --> Data Bit(n) of Address 240h+Y ;  (W=12) (96x64 bits)
Column 12+(n*W) --> Data Bit(n) of Address 260h+Y ;/
```

Cautions: The bits are inverted (0=1, 1=0) in some (not all) dumps. Mind that the bytes are arranged in non-linear polynomial fashion (see PC register).
Recommended format for binary ROM-images would be to undo the inversion (if present), and to maintain the polynomial byte-order.
Note: Known decapped/dumped CICs are D411 and 3195A, and... somebody decapped/dumped a CIC without writing down its part number (probably=6113).

CIC Timings
The NES CICs are driven by a 4.000MHz CIC oscillator (located in the console, and divided by 4 in the NES CIC). The SNES CICs are driven by the 24.576MHz APU oscillator (located and divided by 8 in the console's audio circuit, and further divided by 3 in the SNES CIC).
Ie. internally, the CICs are clocked at 1.000MHz (NES) or 1.024MHz (SNES). All opcodes are executed within 1 clock cycles, except for the 2-byte long jumps (opcodes 78h-7Fh) which take 2 clock cycles. The "skip" opcodes are forcing the following opcode to be executed as a "nop" (ie. the skipped opcode still takes 1 clock cycle; or possibly 2 cycles when skipping long jump opcodes, in case the CPU supports skipping 2-byte opcodes at all).
After Reset gets released, the CICs execute the first opcode after a short delay (3195A: randomly 1.0 or 1.25 cycles, D413A: constantly 1.33 cycles) (whereas, portions of that delay may rely on a poorly falling edge of the incoming Reset signal).

CIC Ports

```
Name  Pin  Dir  Expl
P0.0  1    Out  Data Out    ;\SNES version occassionally swaps these
P0.1  2    In   Data In     ;/pins by software (ie. Pin1=In, Pin2=Out)
P0.2  3    In   Random Seed (0=Charged/Ready, 1=Charging/Busy)
P0.3  4    In   Lock/Key    (0=Cartridge/Key, 1=Console/Lock)
P1.0  9    Out  Reset SNES  (0=Reset Console, 1=No)
P1.1  10   Out  Reset Key   (0=No, 1=Reset Key)
P1.2  11   In   Unused, or Reset Speed A (in 3195A) ;\blink speed of reset
P1.3  12   In   Unused, or Reset Speed B (in 3195A) ;/signal (and Power LED)
P2.0  13   -    Unused
P2.1  14   -    Unused
P2.2  15   -    Unused
P2.3  -    -    Unused
P3.0  -    RAM  Unused, or used as "noswap" flag (in SNES CIC)
P3.1  -    -    Unused
P3.2  -    -    Unused
P3.3  -    -    Unused
```

P0.0-P2.2 are 11 external I/O lines (probably all bidirectional, above directions just indicates how they are normally used). P2.3-P3.3 are 5 internal bits (which seem to be useable as "RAM"). Pin numbers are for 16pin NES/SNES DIP chips (Pin numbers on 18pin SNES SMD chips are slightly rearranged).
P4,P5,P6,P7,P8,P9,PA,PB,PC,PD,PF are unknown/unused (maybe 12x4 further bits, or mirrors of P0..P3).

CIC Stream Seeds
There are different seeds used for different regions. And, confusingly, there is a NES-CIC clone made Tengen, which uses different seeds than the real CIC (some of the differences automatically compensated when summing up values, eg. 8+8 gives same 4bit result as 0+0, other differences are manually adjusted by Tengen's program code).
Many of the reverse-engineered NES seeds found in the internet are based on the Tengen design (the USA-seeds extracted from the decapped Tengen chip, the EUR/ASIA/UK-seeds based on sampled Nintendo-CIC data-streams, and then converted to a Tengen-compatible seed format). To convert them to real CIC seeds:

```
Nintendo[1..F] = Tengen[1..F] - (2,0,0,0,0,0,8,8,8,8,8,8,8,8,2)
```

There are other (working) variations possible, for example:

```
Nintendo[1..F] = Tengen[1..F] - (2,0,0,0,0,0,A,E,8,8,8,8,8,8,8,2)
(That, for Tengen-USA seeds. The Tengen-style-EUR/ASIA/UK seeds may differ)
```

Whereas, the random seed in TengenKEY[1] is meant to be "r+2" (so subtracting 2 restores "r").

CIC Stream Logs
There are some stream logs with filename "XXXX-N.b" where XXXX is the chip name, and N is the random seed, and bytes in the file are as so:

```
Byte 000h, bit0-7 = 1st-8th bit on Pin 1 (DTA.OUT on NES)(DTA.OUT/IN on SNES)
Byte 001h, bit0-7 = 1st-8th bit on Pin 2 (DTA.IN on NES) (DTA.IN/OUT on SNES)
Byte 002h, bit0-7 = 9th-16th bit on Pin 1
Byte 003h, bit0-7 = 9th-16th bit on Pin 2
etc.
```

Caution: The "N" in the filename is taken as if the seed were transferred in order Bit 3,2,1,0 (actually it is Bit 3,0,1,2). Ie. file "3195-1.b" would refer to a NES-EUR-CIC with seed r=4. The signals in the files are sampled at 1MHz (ie. only each fourth 4MHz cycle).

The 6113 Chip
The 6113 chip was invented in 1987, and it replaced the 3193 chip in US/Canadian cartridges (while US/Canadian consoles kept using 3193 chips). When used in cartridges, the 6113 does usually "emulate" a 3193 chip. But, for whatever reason, it can do more:

```
Console  Cartridge  Notes
```

```
  3193     3193       Works (the "old" way)        ;\used combinations
  3193     6113       Works (the "new" way)        ;/
  6113     6113       Works (special seed/timing)  ;\
  6113     3193       Doesn't work                 ; not used as far as known
  6113     ??         Might work (??=unknown chip) ;/
```

When used in consoles, the 6113 uses slightly different timings and seed values (and does request cartridges with 6113 chips to use the 6113-mode, too, rather than emulating the 3193).
One guess: Maybe Nintendo originally used different CICs for NTSC regions (like 3193/3194 for USA/Canada/SouthKorea), and later combined them to one region (if so, all NES consoles in Canada or SouthKorea should contain 3194/6113 chips, unlike US consoles which have 3193 chips).

3195A Signals (NES, Europe)
The I/O ports are HIGH (for Output "1"), or LOW-Z (for Output "0" or Input). Raising edges take circa 0.5us, falling edges take circa 3us.

```
  4MHz Clock Units     ..............................
  1MHz Clock Units       .   .   .   .   .   .   .   .
                          _____________                ;\Console+Cartridge
  Data Should-be       __|             |_____________ ;/should be 3us High
                            _________                  ;\actually 2.5us High
  Data From Console    __.'           ''----.......___ ;/and 3us falling
                          _________                    ;\
  Data From Cartridge  __.'           ''----.......___ ; either same as console
    or, delayed:            _________                ; or 0.25us later
  Data From Cartridge  ___.'           ''----.......__ ;/
```

After Power-up, the Cartridge CIC does randomly start with good timing, or with all signals delayed by 0.25us. In other words, the Cartridge CIC executes the first opcode 1.0us or 1.25us (four or five 4MHz cycles) after Reset gets released. However, for some reason, pushing the Reset Button doesn't alter the timing, the random-decision occurs only on Power-up.

D413A Signals (SNES, Europe)
The D413A signals are looking strange. First, the software switches signals High for 3us, but the actual signals are 3.33us High. Second, the signals on one pin are constantly jumping back'n'forth by 1.33us (in relation to the other pin).

```
  3.072MHz Clock Units .............................
  1.024MHz Clock Units   .  .  .  .  .  .  .  .  .  .
                          ________                     ;\Console+Cartridge
  Data Should-be       ___|        |_____________     ;/should be 3us High
                           _________                   ;\actually 3.33us high
  Data From/To Console ____|         '--...________    ;/and 2us falling
                          _________                    ;\
  Data From/To Cart    ____|         '--...___________ ; 1.33us earlier
   or, delayed                _________               ; or 1.33us later
  Data From/To Cart    ______________|         '--..._ ;/
```

The earlier/later effect occurs because the SNES CICs are occassionally reversing the data-direction of the pins. Ie. in practice, Data from Cartridge is constantly 1.33us LATER than from Console.
Software-wise, the D411 (and probably D413A) is programmed as if the Cartridge CIC would start "immediately", but in practice, it starts 1.33us (four 3.072MHz cycles) after releasing Reset (that offset seems to be constant, unlike as on the 3195A where it randomly changes between 1.0us and 1.25us).

SNES Cartridge CIC Versions

NES CIC Versions

```
  3193,3193A        NES NTSC Cartridges and Consoles       ;\USA,Canada
  6113,6113A,6113B1 NES NTSC Cartridges (not consoles)     ;/(and Korea?)
  3194              Unknown/doesn't exist?
  3195,3193A        NES PAL Cartridges and Consoles "PAL-B";-Europe
  3196(A?)          NES PAL Cartridges and Consoles        ;-Hong Kong,Asia
  3197(A?)          NES PAL Cartridges and Consoles "PAL-A";-UK,Italy,Australia
  3198(A?)          FamicomBox CIC Cartridges and Consoles ;\
  3199(A?)          FamicomBox Coin Timer (not a CIC)      ; Japan
  N/A               Famicom Cartridges and Consoles        ;/
  RFC-CPU10 (?)     NES R.O.B. robot (no CIC, but maybe a 4bit Sharp CPU, too?)
```

SNES CIC Versions

```
  F411,F411A,F411B   SNES NTSC Cartridges-with-SMD-Chipset and Consoles
  D411,D411A,D411B   SNES NTSC Cartridges-with-DIP-Chipset
  F413,F413A,F413B   SNES PAL Cartridges-with-SMD-Chipset and Consoles
  D413,D413A,D413B   SNES PAL Cartridges-with-DIP-Chipset
  SA-1,S-DD1,MCC-BSC SNES Cartridges (coprocessors/mappers with on-chip CIC)
```

NES CIC Clones

```
  23C1033
  337002   ;Tengen's 16pin "Rabbit" CIC clone
  337006   ;Tengen's 40pin "RAMBO-1" mapper with built-in CIC clone
  4051
  7660
  KC5373B
  MX8018
  NINA
  Ciclone  ;homebrew multi-region CIC clone (based on Tengen design)
```

Aside from using cloned CICs, many unlicensed NES cartridges used a different approach: injecting "wrong" voltages to the console, and "stunning" its CIC.

SNES CIC Clones

```
  10198    - CIC clone
  noname   - CIC clone (black chip without any part number)
  ST10198S - NTSC CIC clone
  ST10198P - PAL CIC clone
  265111   - maybe also a CIC clone (used in Bung Game Doctor SF6)
  D1       - maybe also a CIC clone (used in Super UFO Pro8)
  74LS112  - reportedly also a CIC clone (with fake part number) (UFO Pro6)
  CIVIC 74LS13   16pin - CIC/D411 clone (used in a 8-in-1 pirate cart)
  CIVIC CT6911   16pin - CIC      clone (used in a 7-in-1 pirate cart)
  93C26          16pin - CIC      clone (used in a 8-in-1 pirate cart)
  D1             16pin - CIC? (used in Super VG pirate)
  STS9311A 52583 16pin - CIC clone (used in Donkey King Country 3 pirate)
  black blob     16pin - CIC/D411 clone (used in Sonic the Hedgehog pirate)
```

CIC Chip Year/Week Date Codes

```
  Name   YYWW-YYWW
```

```
3193   8539-8642
3193A  8547-8733 (in cartridges) (but should be in consoles for more years)
3195   8627-8638
3195A  8647-9512
3197A  8647-9227
6113   8734-8823
6113A  8823-8933
6113B1 8847-9344
```

SNES Cart LoROM Mapping (ROM divided into 32K banks) (around 1500 games)

Plain LoROM

Board Type	ROM Area	ROM Mirrors
SHVC-1A0N-01,02,10,20,30	00-7D,80-FF:8000-FFFF	40-7D,C0-FF:0000-7FFF
SHVC-2A0N-01,10,11,20	00-7D,80-FF:8000-FFFF	40-7D,C0-FF:0000-7FFF
SHVC-BA0N-01,10	00-7D,80-FF:8000-FFFF	40-7D,C0-FF:0000-7FFF
SHVC-YA0N-01	00-7D,80-FF:8000-FFFF	40-7D,C0-FF:0000-7FFF

LoROM with SRAM

Board Type	ROM Area	SRAM Area
SHVC-1A1B-04,05,06	00-1F,80-9F:8000-FFFF	70-7D,F0-FF:0000-FFFF
SHVC-1A3B-11,12,13	00-1F,80-9F:8000-FFFF	70-7D,F0-FF:0000-FFFF
SHVC-1A5B-02,04	00-1F,80-9F:8000-FFFF	70-7D,F0-FF:0000-FFFF
SHVC-2A3B-01	00-3F,80-BF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-2A3M-01 with MAD-R	00-3F,80-BF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-2A3M-01,11,20	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-1A3B-20	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-1A1M-01,11,20	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-2A1M-01	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-BA1M-01	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-1A3M-10,20,21,30	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-BA3M-01	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-1A5M-01,11,20	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-2A5M-01	00-7D,80-FF:8000-FFFF	70-7D,F0-FF:0000-7FFF
SHVC-1A7M-01	?	?

Note that 2A3M-01 exists with/without MAD-R (and have different mappings).
Note that 1A3B-20 differs from earlier 1A3B-xx versions.
The older boards map SRAM to the whole 64K areas at banks 70h-7Dh/F0-FFh.
The newer boards map SRAM to the lower 32K areas at banks 70h-7Dh/F0-FFh (this allows "BigLoROM" games to use the upper 32K of that banks as additional LoROM banks, which is required for games with more than 3MB LoROM).
Most of the existing boards contain 0K, 2K, 8K, or 32K SRAM. A few games contail 64K or 128K SRAM, which is divided into 32K chunks, mapped to bank 70h, 71h, etc.)

Some LoROM games are bigger than 2Mbytes (eg. Super Metroid, Gunple, Wizardry 6, Derby Stallion 3), these have bank 0-3Fh mapped in the 32K LoROM banks as usually, and bank 40h and up each mapped twice in the 64K hirom banks.
Note: There's also a different "SpecialLoROM" mapping scheme for 3MByte ROMs (used by Derby Stallion 96 and Sound Novel Tsukuru; aside from the special ROM mapping, these cartridges have an additional Data Pack Slot).

SNES Cart HiROM Mapping (ROM divided into 64K banks) (around 500 games)

Plain HiROM

Board Type	ROM Area at 0000-FFFF	ROM Mirrors at 8000-FFFF	SRAM Area (none such)
SHVC-BJ0N-01,20	40-7d,c0-ff	00-3f,80-bf	N/A
SHVC-YJ0N-01	40-7d,c0-ff	00-3f,80-bf	N/A
SHVC-1J0N-01,10,20	40-7d,c0-ff	00-3f,80-bf	N/A
SHVC-2J0N-01,10,11	40-7d,c0-ff	00-3f,80-bf	N/A
SHVC-3J0N-01	40-6f,c0-ef	00-2f,80-af	N/A

The SHVC-3J0N-01 board contains 3 ROM chips (memory is divided into chunks of 16 banks, with one ROM per chunk, and with each 4th chunk being left empty, ie. bank 30-3F,70-7D,B0-BF,F0-FF are open-bus).

HiROM with SRAM

Board Type	ROM Area at 0000-FFFF	ROM Mirrors at 8000-FFFF	SRAM Area at 6000-7FFF
SHVC-1J3B-01	40-7d,c0-ff	00-3f,80-bf	20-3f,a0-bf
SHVC-1J1M-11,20	40-7d,c0-ff	00-3f,80-bf	20-3f,a0-bf
SHVC-1J3M-01,11,20	40-7d,c0-ff	00-3f,80-bf	20-3f,a0-bf
SHVC-BJ3M-10	40-7d,c0-ff	00-3f,80-bf	20-3f,a0-bf
SHVC-1J5M-11,20	40-7d,c0-ff	00-3f,80-bf	20-3f,a0-bf
SHVC-2J3M-01,11,20	40-7d,c0-ff	00-3f,80-bf	10-1f,30-3f,90-9f,b0-bf
SHVC-2J5M-01	40-7d,c0-ff	00-3f,80-bf	10-1f,90-9f,30-3f,b0-bf
SHVC-LJ3M-01	40-7d,c0-ff	00-3f,80-bf	80-bf

The SHVC-LJ3M-01 board uses ExHiROM mapping (meaning that bank 00h-7Dh contain different ROM banks than 80h-FFh).

SNES Cart SA-1 (programmable 65C816 CPU) (aka Super Accelerator) (35 games)

SNES Cart SA-1 Games
SNES Cart SA-1 I/O Map

SNES Cart SA-1 Interrupt/Control on SNES Side
SNES Cart SA-1 Interrupt/Control on SA-1 Side
SNES Cart SA-1 Timer
SNES Cart SA-1 Memory Control
SNES Cart SA-1 DMA Transfers
SNES Cart SA-1 Character Conversion
SNES Cart SA-1 Arithmetic Maths
SNES Cart SA-1 Variable-Length Bit Processing

SNES Pinouts SA1 Chip

Memory Map (SNES Side)
```
  00h-3Fh/80h-BFh:2200h-23FFh  I/O Ports
  00h-3Fh/80h-BFh:3000h-37FFh  I-RAM (2Kbytes, on-chip, 10MHz fast RAM)
  00h-3Fh/80h-BFh:6000h-7FFFh  One mappable 8Kbyte BW-RAM block
  00h-3Fh/80h-BFh:8000h-FFFFh  Four mappable 1MByte LoROM blocks (max 8Mbyte)
  40h-4Fh:0000h-FFFFh          Entire 256Kbyte BW-RAM (mirrors in 44h-4Fh)
  C0h-FFh:0000h-FFFFh          Four mappable 1MByte HiROM blocks (max 8Mbyte)
```
The SA-1 supports both LoROM and HiROM mappings (eg. LoROM banks 00h-01h mirror to HiROM bank 40h). Default exception vectors (and cartridge header) are always in LoROM bank 00h (ie. at ROM offset 7Fxxh).

Memory Map (SA-1 Side)
Same as on SNES Side (of course without access to SNES internal WRAM and I/O ports), plus following additional areas:
```
  00h-3Fh/80h-BFh:0000h-07FFh  I-RAM (at both 0000h-07FFh and 3000h-37FFh)
  60h-6Fh:0000h-FFFFh          BW-RAM mapped as 2bit or 4bit pixel buffer
```
Some other differences to SNES Side are: I/O Ports are different, on SA-1 side, the mappable BW-RAM area (at 6000h-7FFFh) can be also assigned as 2bit/4bit pixel buffer (on SNES Side it's always normal 8bit memory).

Misc
65C816 CPU at 10.74MHz
```
  2Kbytes internal I-RAM (work ram/stack) (optionally battery backed)
  Optional external backup/work BW-RAM up to 2MByte (or rather only 2Mbit?)
  Addressable ROM up to 8MByte (64MBits)
```

The SA-1 CPU can access memory at 10.74MHz rate (or less, if the SNES does simultaneouly access cartridge memory).
The SNES CPU can access memory at 2.68MHz rate (or 3.5MHz, but that mode may not be used in combination with the SA-1).

When interrupts are disabled (in CIE/SIE), then it sounds as if the interrupt flags still do get set?

"BW-RAM cannot be used during character conversion DMA."

IRQ/NMI/Reset vectors can be mapped. Other vectors (BRK/COP etc) are always taken from ROM (for BOTH CPUs).

```
    XXX pg 62..66 timings
 ok XXX pg 67..78 char/bitmap
 ok XXX pg 79..81 arit
    XXX pg 82..86 var-len
 ok XXX pg 87..90 dma
```

SA-1 Pinouts
```
  1-126  Unknown
  127    PAL/NTSC (for CIC mode and/or HV-timer?)
  128    Unknown
```

SA-1 PCBs
```
  BSC-1L3B-01    NTSC SRAM Battery FLASH-Slot (Itoi Shig. no Bass Tsuri No.1)
  SHVC-1L0N3S-20 NTSC SRAM NoBattery (Dragon Ball Z Hyper Dimension)
  SHVC-1L3B-11   NTSC SRAM Battery
  SHVC-1L5B-10   NTSC SRAM Battery
  SHVC-1L5B-11   NTSC SRAM Battery
  SHVC-1L8B-10   NTSC SRAM Battery
  SNSP-1L0N3S-01 PAL  SRAM NoBattery (Dragon Ball Z Hyper Dimension)
  SNSP-1L3B-20   PAL  SRAM Battery
```
The battery can be wired to I-RAM (on-chip SA-1 memory) or BW-RAM (aka SRAM) or both; unknown how it is wired in practice (probably to BW-RAM?).

Chipset/Components
```
  U1  44pin  ROM (probably with full 16bit databus connected)
  U2  28pin  SRAM (LH52A64N-YL or LH52256ANZ or 32pin LH52A512NF)
  U3  128pin SA1 (SA1 RF5A123)
  U4  8pin   Battery controller MM1026AF  ;\only if PCB does include a battery
  BATT 2pin  CR2032                       ;/
  CN1 62pin  SNES cartridge edge-connector
  CN2 62pin  Satellaview FLASH cartridge slot  ;-only on BSC-boards
```

SNES Cart SA-1 Games

SA1 - 128pin - Super Accelerator (book2) (10.74MHz 65C816 CPU)
Used by 35 games:
```
 #Asahi Shinbun Rensai Kato Ichi-Ni-San Kudan Shogi Shingiru (1995) Varie (JP)
  Daisenryaku Expert WWII: War in Europe (1996) SystemSoftAlpha/ASCII Corp (JP)
  Derby Jockey 2 (1995) Muse Soft/Asmik (JP)
  Dragon Ball Z: Hyper Dimension (1996) TOSE/Bandai (JP) (EU)
 #Habu Meijin no Omoshiro Syouhi -Unverified (19xx) Hiroshi/etc. (JP)
  Itoi Shigesato no Bass Tsuri No. 1 (1997) HAL Laboratory/Nintendo (JP)
  J. League '96 Dream Stadium (1996) Hudson Soft (JP)
  Jikkyou Oshaberi Parodius (1995) Konami (JP)
  Jumpin' Derby (1996) Naxat Soft (JP)
 #Kakinoki Shogi (1995) ASCII Corporation (JP)
  Kirby Super Star (1996) HAL Laboratory/Nintendo (NA) (JP) (EU)
  Kirby's Dream Land 3 (1997) HAL Laboratory/Nintendo (NA) (JP)
  Marvelous: Mouhitotsu no Takarajima (1996) Nintendo/R&D2 (JP)
  Masoukishin: Super Robot Wars Gaiden: Lord of Elemental (19xx) Banpresto (JP)
  Masters New: Haruka Naru Augusta 3 (1995) T&E Soft (JP)
  Mini Yonku/4WD Shining Scorpion - Let's & Go!! (1996) KID/ASCII Corp (JP)
  Pachi Slot Monogatari PAL Kogyo Special -Unverified (1995) PAL/KSS (JP)
  Pebble Beach no Hotou: New Tournament Edition (1996) T&E Soft (JP)
  PGA European Tour (1996) Halestorm/THQ/Black Pearl Software (NA)
  PGA Tour '96 (1995) Black Pearl Software/Electronic Arts (NA)
  Power Rangers Zeo: Battle Racers (1996) Natsume/Bandai (NA)
 #Pro Kishi Simulation Kishi No Hanamichi (1996) Atlus (JP)
 xRin Kaihou 9 Dan No Igo Taidou -Unverified (1996) .. (JP)
  SD F-1 Grand Prix (and "Sample" version) (1995) Video System (JP)
  SD Gundam G NEXT (1995) BEC/Bandai (JP)
```

```
 #Shin Shogi/Syogi Club (1995) Hect/Natsu (JP)
 #Shogi Saikyou (1995) Magical Company (JP) (unverified?)
 #Shogi Saikyou 2 (1996) Magical Company (JP)
 #Shougi Mahjong (1995) Varie Corp (JP)
  Super Bomberman Panic Bomber World (1995) Hudson Soft (JP)
  Super Mario RPG: Legend of the Seven Stars (1996) Square/Nintendo (NA) (JP)
  Super Robot T.G.: The Lord Of Elemental (?) (1996) Winkysoft/Banpresto (JP)
 #Super Shogi 3 Kitaihei -Unverified (1995) I'Max (JP)
 xTaikyoku-Igo Idaten -Unverified (1995) BPS (JP)
 xTakemiya Masaki Kudan No Igo Taisyou -Unverified (1995) KSS (JP)
```

The nine Shogi/Shougi/Syouhi/Kishi/Syogi titles are japanese Chess games, the three Igo titles are Go games; that 12 titles are mainly using the SA-1 CPU for calculating moves, without doing any impressive things with the SA-1 I/O ports.

SNES Cart SA-1 I/O Map

SA-1 I/O Map (Write Only Registers)

```
  Port  Side  Name  Reset Expl.
  2200h SNES  CCNT  20h   SA-1 CPU Control (W)
  2201h SNES  SIE   00h   SNES CPU Int Enable (W)
  2202h SNES  SIC   00h   SNES CPU Int Clear  (W)
  2203h SNES  CRV   -     SA-1 CPU Reset Vector Lsb (W)
  2204h SNES  CRV   -     SA-1 CPU Reset Vector Msb (W)
  2205h SNES  CNV   -     SA-1 CPU NMI Vector Lsb (W)
  2206h SNES  CNV   -     SA-1 CPU NMI Vector Msb (W)
  2207h SNES  CIV   -     SA-1 CPU IRQ Vector Lsb (W)
  2208h SNES  CIV   -     SA-1 CPU IRQ Vector Msb (W)
  2209h SA-1  SCNT  00h   SNES CPU Control (W)
  220Ah SA-1  CIE   00h   SA-1 CPU Int Enable (W)
  220Bh SA-1  CIC   00h   SA-1 CPU Int Clear  (W)
  220Ch SA-1  SNV   -     SNES CPU NMI Vector Lsb (W)
  220Dh SA-1  SNV   -     SNES CPU NMI Vector Msb (W)
  220Eh SA-1  SIV   -     SNES CPU IRQ Vector Lsb (W)
  220Fh SA-1  SIV   -     SNES CPU IRQ Vector Msb (W)
  2210h SA-1  TMC   00h   H/V Timer Control (W)
  2211h SA-1  CTR   -     SA-1 CPU Timer Restart (W)
  2212h SA-1  HCNT  -     Set H-Count Lsb (W)
  2213h SA-1  HCNT  -     Set H-Count Msb (W)
  2214h SA-1  VCNT  -     Set V-Count Lsb (W)
  2215h SA-1  VCNT  -     Set V-Count Msb (W)
  2216h -     -     -     -
  2220h SNES  CXB   00h   MMC Bank C - Hirom C0h-CFh / LoRom 00h-1Fh (W)
  2221h SNES  DXB   01h   MMC Bank D - Hirom D0h-DFh / LoRom 20h-3Fh (W)
  2222h SNES  EXB   02h   MMC Bank E - Hirom E0h-EFh / LoRom 80h-9Fh (W)
  2223h SNES  FXB   03h   MMC Bank F - Hirom F0h-FFh / LoRom A0h-BFh (W)
  2224h SNES  BMAPS 00h   SNES CPU BW-RAM Mapping to 6000h-7FFFh (W)
  2225h SA-1  BMAP  00h   SA-1 CPU BW-RAM Mapping to 6000h-7FFFh (W)
  2226h SNES  SBWE  00h   SNES CPU BW-RAM Write Enable (W)
  2227h SA-1  CBWE  00h   SA-1 CPU BW-RAM Write Enable (W)
  2228h SNES  BWPA  FFh   BW-RAM Write-Protected Area (W)
  2229h SNES  SIWP  00h   SNES I-RAM Write-Protection (W)
  222Ah SA-1  CIWP  00h   SA-1 I-RAM Write-Protection (W)
  222Bh -     -     -     -
  2230h SA-1  DCNT  00h   DMA Control (W)
  2231h Both  CDMA  00h   Character Conversion DMA Parameters (W)
  2232h Both  SDA   -     DMA Source Device Start Address Lsb (W)
  2233h Both  SDA   -     DMA Source Device Start Address Mid (W)
  2234h Both  SDA   -     DMA Source Device Start Address Msb (W)
  2235h Both  DDA   -     DMA Dest Device Start Address Lsb (W)
  2236h Both  DDA   -     DMA Dest Device Start Address Mid (Start/I-RAM) (W)
  2237h Both  DDA   -     DMA Dest Device Start Address Msb (Start/BW-RAM)(W)
  2238h SA-1  DTC   -     DMA Terminal Counter Lsb (W)
  2239h SA-1  DTC   -     DMA Terminal Counter Msb (W)
  223Ah -     -     -     -
  223Fh SA-1  BBF   00h   BW-RAM Bit Map Format for 600000h-6FFFFFh (W)
  224xh SA-1  BRF   -     Bit Map Register File (2240h..224Fh) (W)
  2250h SA-1  MCNT  00h   Arithmetic Control (W)
  2251h SA-1  MA    -     Arithmetic Param A Lsb (Multiplicand/Dividend) (W)
  2252h SA-1  MA    -     Arithmetic Param A Msb (Multiplicand/Dividend) (W)
  2253h SA-1  MB    -     Arithmetic Param B Lsb (Multiplier/Divisor) (W)
  2254h SA-1  MB    -     Arithmetic Param B Msb (Multiplier/Divisor)/Start (W)
  2255h -     -     -     -
  2258h SA-1  VBD   -     Variable-Length Bit Processing (W)
  2259h SA-1  VDA   -     Var-Length Bit Game Pak ROM Start Address Lsb (W)
  225Ah SA-1  VDA   -     Var-Length Bit Game Pak ROM Start Address Mid (W)
  225Bh SA-1  VDA   -     Var-Length Bit Game Pak ROM Start Address Msb & Kick
  225Ch -     -     -     -
  2261h -     -     -     Unknown/Undocumented (Jumpin Derby writes 00h)
  2262h -     -     -     Unknown/Undocumented (Super Bomberman writes 00h)
```

SA-1 I/O Map (Read Only Registers)

```
  Port  Side  Name  Reset Expl.
  2300h SNES  SFR   SNES CPU Flag Read (R)
  2301h SA-1  CFR   SA-1 CPU Flag Read (R)
  2302h SA-1  HCR   H-Count Read Lsb / Do Latching (R)
  2303h SA-1  HCR   H-Count Read Msb (R)
  2304h SA-1  VCR   V-Count Read Lsb (R)
  2305h SA-1  VCR   V-Count Read Msb (R)
  2306h SA-1  MR    Arithmetic Result, bit0-7   (Sum/Product/Quotient) (R)
  2307h SA-1  MR    Arithmetic Result, bit8-15  (Sum/Product/Quotient) (R)
  2308h SA-1  MR    Arithmetic Result, bit16-23 (Sum/Product/Remainder) (R)
  2309h SA-1  MR    Arithmetic Result, bit24-31 (Sum/Product/Remainder) (R)
  230Ah SA-1  MR    Arithmetic Result, bit32-39 (Sum) (R)
  230Bh SA-1  OF    Arithmetic Overflow Flag (R)
  230Ch SA-1  VDP   Variable-Length Data Read Port Lsb (R)
  230Dh SA-1  VDP   Variable-Length Data Read Port Msb (R)
  230Eh SNES  VC    Version Code Register (R)
```

Reset

Port 2200h = 20h. Port 2228h = FFh. Ports 2220h-2223h = 00h,01h,02h,03h. Ports 2201h-2202h, 2209h-220Bh, 2210h, 2224h-2227h, 2229h-222Ah, 2230h-2231h, 223Fh, 2250h = 00h. Ports 2203h-2208h, 220Ch-220Fh, 2211h-2215h, 2232h-2239h, 2240h-224Fh, 2251h-2254h, 2258h-225Bh = N/A.

SNES Cart SA-1 Interrupt/Control on SNES Side

2200h SNES CCNT - SA-1 CPU Control (W)
```
0-3 Message from SNES to SA-1 (4bit value)
4   NMI from SNES to SA-1   (0=No Change?, 1=Interrupt)
5   Reset from SNES to SA-1 (0=No Reset, 1=Reset)
6   Wait from SNES to SA-1  (0=No Wait, 1=Wait)
7   IRQ from SNES to SA-1   (0=No Change?, 1=Interrupt)
```
Unknown if Wait freezes the whole SA1 (CPU, plus Timer and DMA?).
Unknown if Reset resets any I/O Ports (such like DMA or interrupts) or if it does only reset the CPU?

2201h SNES SIE - SNES CPU Int Enable (W)
```
0-4 Not used (should be 0)
5   IRQ Enable (Character conversion DMA) (0=Disable, 1=Enable)
6   Not used (should be 0)
7   IRQ Enable (from SA-1) (0=Disable, 1=Enable)
```

2202h SNES SIC - SNES CPU Int Clear (W)
```
0-4 Not used (should be 0)
5   IRQ Acknowledge (Character conversion DMA) (0=No change, 1=Clear)
6   Not used (should be 0)
7   IRQ Acknowledge (from SA-1) (0=No change, 1=Clear)
```

2203h SNES CRV - SA-1 CPU Reset Vector Lsb (W)
2204h SNES CRV - SA-1 CPU Reset Vector Msb (W)
2205h SNES CNV - SA-1 CPU NMI Vector Lsb (W)
2206h SNES CNV - SA-1 CPU NMI Vector Msb (W)
2207h SNES CIV - SA-1 CPU IRQ Vector Lsb (W)
2208h SNES CIV - SA-1 CPU IRQ Vector Msb (W)
Exception Vectors on SA-1 side (these are ALWAYS replacing the normal vectors in ROM).

2300h SNES SFR - SNES CPU Flag Read (R)
```
0-3 Message from SA-1 to SNES (4bit value)          (same as 2209h.Bit0-3)
4   NMI Vector for SNES (0=ROM FFExh, 1=Port 220Ch) (same as 2209h.Bit4)
5   IRQ from Character Conversion DMA (0=None, 1=Interrupt) (ready-to-do-DMA)
6   IRQ Vector for SNES (0=ROM FFExh, 1=Port 220Eh) (same as 2209h.Bit6)
7   IRQ from SA-1 to SNES   (0=None, 1=Interrupt) (triggered by 2209h.Bit7)
```
Bit0-3,4,6 are same as in Port 2209h. Bit5 is set via ..DMA..? Bit7 is set via Port 2209h. Bit5,7 can be cleared via Port 2202h.

230Eh SNES VC - Version Code Register (R)
```
0-7  SA-1 Chip Version
```
Existing value(s) are unknown. There seems to be only one chip version (labeled SA-1 RF5A123, used for both PAL and NTSC). The "VC" register isn't read by any games (except, accidently, by a bugged memcopy function at 059E92h in Derby Jockey 2).

SNES Cart SA-1 Interrupt/Control on SA-1 Side

2209h SA-1 SCNT - SNES CPU Control (W)
```
0-3 Message from SA-1 to SNES (4bit value)
4   NMI Vector for SNES (0=ROM FFEAh, 1=Port 220Ch)
5   Not used (should be 0)
6   IRQ Vector for SNES (0=ROM FFEEh, 1=Port 220Eh)
7   IRQ from SA-1 to SNES   (0=No Change?, 1=Interrupt)
```

220Ah SA-1 CIE - SA-1 CPU Int Enable (W)
```
0-3 Not used (should be 0)
4   NMI Enable (from SNES)  (0=Disable, 1=Enable)
5   IRQ Enable (from DMA)   (0=Disable, 1=Enable)
6   IRQ Enable (from Timer) (0=Disable, 1=Enable)
7   IRQ Enable (from SNES)  (0=Disable, 1=Enable)
```

220Bh SA-1 CIC - SA-1 CPU Int Clear (W)
```
0-3 Not used (should be 0)
4   NMI Acknowledge (from SNES)  (0=No change, 1=Clear)
5   IRQ Acknowledge (from DMA)   (0=No change, 1=Clear)
6   IRQ Acknowledge (from Timer) (0=No change, 1=Clear)
7   IRQ Acknowledge (from SNES)  (0=No change, 1=Clear)
```

220Ch SA-1 SNV - SNES CPU NMI Vector Lsb (W)
220Dh SA-1 SNV - SNES CPU NMI Vector Msb (W)
220Eh SA-1 SIV - SNES CPU IRQ Vector Lsb (W)
220Fh SA-1 SIV - SNES CPU IRQ Vector Msb (W)
Exception Vectors on SNES side (these are optionally replacing the normal vectors in ROM; depending on bits in Port 2209h; the "I/O" vectors are used only by Jumpin Derby, all other games are using the normal ROM vectors).

2301h SA-1 CFR - SA-1 CPU Flag Read (R)
```
0-3 Message from SNES to SA-1 (4bit value)       (same as 2200h.Bit0-3)
4   NMI from SNES to SA-1   (0=No, 1=Interrupt)  (triggered by 2200h.Bit4)
5   IRQ from DMA to SA-1    (0=No, 1=Interrupt)  (triggered by DMA-finished)
6   IRQ from Timer to SA-1  (0=No, 1=Interrupt)  (triggered by Timer)
7   IRQ from SNES to SA-1   (0=No, 1=Interrupt)  (triggered by 2200h.Bit7)
```

SNES Cart SA-1 Timer

2210h SA-1 TMC - H/V Timer Control (W)

```
  0   HEN             ;\Enables Interrupt or so ?
  1   VEN             ;/
  2-6 Not used (should be 0)
  7   Timer Mode (0=HV Timer, 1=Linear Timer)
```

2211h SA-1 CTR - SA-1 CPU Timer Restart (W)
```
  0-7 Don't care (writing any value restarts the timer at 0)
```

2212h SA-1 HCNT - Set H-Count Lsb (W)
2213h SA-1 HCNT - Set H-Count Msb (W)
```
  0-8  H-Counter (9bit)
  9-15 Not used (should be 0)
```
Ranges from 0-340 (in HV mode), or 0-511 (in Linear mode).

2214h SA-1 VCNT - Set V-Count Lsb (W)
2215h SA-1 VCNT - Set V-Count Msb (W)
```
  0-8  V-Counter (9bit)
  9-15 Not used (should be 0)
```
Ranges from 0-261 (in HV/NTSC mode), 0-311 (in HV/PAL mode), or 0-511 (in Linear mode). The PAL/NTSC selection is probably done by a soldering point on the PCB (which is probably also used for switching the built-in CIC to PAL/NTSC mode).

2302h SA-1 HCR - H-Count Read Lsb / Do Latching (R)
2303h SA-1 HCR - H-Count Read Msb (R)
2304h SA-1 VCR - V-Count Read Lsb (R)
2305h SA-1 VCR - V-Count Read Msb (R)
Reading from 2302h automatically latches the other HV-Counter bits to 2303h-2305h.

Notes
In HV-mode, the timer clock is obviously equivalent to the dotclock (four 21MHz master cycles per dot). The time clock in linear mode is unknown (probably same as in HV-mode).
H-counter has 341 dots (one more as in SNES, but without long dots). Unknown if the short-scanline (in each 2nd NTSC non-interlaced frame) is reproduced (if it isn't, then one must periodically reset the timer in order to keep it in sync with the PPU). There is no provision for interlaced video timings.
The meaning of Port 2212h-2215h is totally unknown (according to existing specs it <sounds> as if they do set the <current> counter value - though alltogether it'd be more likely that they do contain <compare> values).
Unknown what happens when setting both HEN and VEN (probably IRQ triggers only if <both> H+V do match, ie. similar as for the normal SNES timers).

SNES Cart SA-1 Memory Control

2220h SNES CXB - Set Super MMC Bank C - Hirom C0h-CFh / LoRom 00h-1Fh (W)
2221h SNES DXB - Set Super MMC Bank D - Hirom D0h-DFh / LoRom 20h-3Fh (W)
2222h SNES EXB - Set Super MMC Bank E - Hirom E0h-EFh / LoRom 80h-9Fh (W)
2223h SNES FXB - Set Super MMC Bank F - Hirom F0h-FFh / LoRom A0h-BFh (W)
```
  0-2  Select 1Mbyte ROM-Bank (0..7)
  3-6  Not used (should be 0)
  7    Map 1Mbyte ROM-Bank (0=To HiRom, 1=To LoRom and HiRom)
```
If LoRom mapping is disabled (bit7=0), then first 2 MByte of ROM are mapped to 00h-3Fh, and next 2 MByte to 80h-BFh. The registers do affect both SNES and SA-1 mapping.

2224h SNES BMAPS - SNES CPU BW-RAM Mapping to 6000h-7FFFh (W)
```
  0-4  Select 8Kbyte BW-RAM Block for mapping to 6000h-7FFFh (0..31)
  5-7  Not used (should be 0)
```
BW-RAM is always mapped to bank 40h-43h (max 256 Kbytes).
This register allows to map an 8Kbyte chunk to offset 6000h-7FFFh in bank 0-3Fh and 80h-BFh.

2225h SA-1 BMAP - SA-1 CPU BW-RAM Mapping to 6000h-7FFFh (W)
```
  0-6  Select 8Kbyte BW-RAM Block for mapping to 6000h-7FFFh (0..31 or 0..127)
  7    Select source (0=Normal/Bank 40h..43h, 1=Bitmap/Bank 60h..6Fh)
```

223Fh SA-1 BBF - BW-RAM Bit Map Format for 600000h-6FFFFFh (W)
```
  0-6 Not used (should be "..") (whatever ".." means, maybe "0"?)
  7   Format (0=4bit, 1=2bit)
```
"BW-RAM bitmap logical space format setting from perspective of the SA-1 CPU"
```
  600000h.Bit0-1 or Bit0-3 mirrors to 400000h.Bit0-1 or 400000h.Bit0-3
  600001h.Bit0-1 or Bit0-3 mirrors to 400000h.Bit2-3 or 400000h.Bit4-7
  600002h.Bit0-1 or Bit0-3 mirrors to 400000h.Bit4-5 or 400001h.Bit0-3
  600003h.Bit0-1 or Bit0-3 mirrors to 400000h.Bit6-7 or 400001h.Bit4-7
  etc.
```
Note that the LSBs in the packed-area contain the left-most pixel (not the right-most one). The MSBs in the unpacked area are "ignored" (this is obvious in case of writing; for reading it's unknown what it means - are reads supported at all, and if so, do they return zero's or garbage in MSBs?)

2226h SNES SBWE - SNES CPU BW-RAM Write Enable (W)
2227h SA-1 SBWE - SA-1 CPU BW-RAM Write Enable (W)
```
  0-6  Not used (should be 0)
  7    Write Enable BW-RAM (0=Protect, 1=Write Enable)
```

2228h SNES BWPA - BW-RAM Write-Protected Area (W)
```
  0-3  Select size of Write-Protected Area ("256 SHL N" bytes)
  4-7  Not used (should be 0)
```
Selects how many bytes (originated at 400000h) shall be write protected.
It isn't possible to set the size to "none" (min is 256 bytes), though, one can probably completely disable the protection via ports 2226h/2227h?

2229h SNES SIWP - SNES I-RAM Write-Protection (W)
222Ah SA-1 CIWP - SA-1 I-RAM Write-Protection (W)
```
  0-7  Write enable flags for eight 256-byte chunks (0=Protect, 1=Write Enable)
```
Bit0 for I-RAM 3000h..30FFh, bit1 for 3100h..31FFh, etc. bit7 for 3700h..37FFh.

SNES Cart SA-1 DMA Transfers

2230h SA-1 DCNT - DMA Control (W)
```
0-1 DMA Source Device      (0=ROM, 1=BW-RAM, 2=I-RAM, 3=Reserved);\for
2   DMA Destination Device (0=I-RAM, 1=BW-RAM)                  ;/Normal DMA
3   Not used (should be 0)
4   DMA Char Conversion Type (0=Type 2/Semi-Automatic, 1=Type 1/Automatic)
5   DMA Char Conversion Enable (0=Normal DMA, 1=Character Conversion DMA)
6   DMA Priority (0=SA-1 CPU Priority, 1=DMA Priority) ;<-- for Normal DMA
7   DMA Enable (0=Disable, 1=Enable... and Clear Parameters?)
```
Bit6 is only valid for Normal DMA between BW-RAM and I-RAM. Source and Destination may not be the same devices (ie. no I-RAM to I-RAM, or BW-RAM to BW-RAM).

2231h Both CDMA - Character Conversion DMA Parameters (W)
```
0-1 Color Depth (0=8bit, 1=4bit, 2=2bit, 3=Reserved)
2-4 Virtual VRAM Width (0..5 = 1,2,4,8,16,32 characters) (6..7=Reserved)
5-6 Not used (should be 0)
7   Terminate Character Conversion 1 (0=No change, 1=Terminate DMA)
```

2232h Both SDA - DMA Source Device Start Address Lsb (W)
2233h Both SDA - DMA Source Device Start Address Mid (W)
2234h Both SDA - DMA Source Device Start Address Msb (W)
```
0-23  24bit Memory Address (translated to 23bit ROM Offset via 2220h..2223h)
0-17  18bit BW-RAM Offset
0-10  11bit I-RAM Offset
```
Used bits are 24bit/18bit/11bit for ROM/BW-RAM/I-RAM.

2235h Both DDA - DMA Destination Device Start Address Lsb (W)
2236h Both DDA - DMA Destination Device Start Address Mid (Start/I-RAM) (W)
2237h Both DDA - DMA Destination Device Start Address Msb (Start/BW-RAM)(W)
```
0-17  BW-RAM Offset (transfer starts after writing 2237h)
0-10  I-RAM Offset  (transfer starts after writing 2236h) (2237h is unused)
```

2238h SA-1 DTC - DMA Terminal Counter Lsb (W)
2239h SA-1 DTC - DMA Terminal Counter Msb (W)
```
0-15  DMA Transfer Length in bytes (1..65535) (0=Reserved/unknown)
```
DTC is used only for Normal DMA (whilst Character Conversion DMA lasts endless; for Type 1: as long as SNES reads "BW-RAM" / until it sets 2231h.Bit7, for Type 2: as long as SA-1 writes BRF / until it clears 2230h.Bit0).

224xh SA-1 BRF - Bit Map Register File (2240h..224Fh) (W)
These 16 registers can hold two 8 pixel rows (with 2bit/4bit/8bit per pixel).
```
0-1  2bit pixel (bit 2-7=unused)
0-3  4bit pixel (bit 4-7=unused)
0-7  8bit pixel
```
Used only for (semi-automatic) Character Conversion Type 2, where the "DMA" source data is to be written pixel-by-pixel to these registers; writing to one 8 pixel row can be done while transferring the other row to the SNES.

Normal DMA (memory transfer within cartridge memory)
```
ROM    --> I-RAM     10.74MHz
ROM    --> BW-RAM    5.37MHz
BW-RAM --> I-RAM     5.37MHz
I-RAM  --> BW-RAM    5.37MHz
```
For normal DMA:
```
Set DCNT (select source/dest/prio/enable)
Set SDA (set source offset)
Set DTC (set transfer length)
Set DDA (set destination offset, and start transfer)
If desired, wait for CFR.Bit5 (DMA completion interrupt)
```
Normal DMA is used by J. League '96, Jumpin Derby, Marvelous. For ROM, SDA should be usually C00000h and up (HiROM mapping); Jumpin Derby is unconventionally using SDA at 2x8xxxh and up (LoROM mapping).

Character Conversion DMA
Used to convert bitmaps or pixels to bit-planed tiles. For details, see
SNES Cart SA-1 Character Conversion

SNES DMA (via Port 43xxh)
Can be used to transfer "normal" data from ROM/BW-RAM/I-RAM to SNES memory, also used for forwarding temporary Character Conversion data from I-RAM to SNES.

Unknown details
Unknown if SDA/DDA are increased and if DTC is decreased (or if that operations appear only on internal registers) (MSBs of DDA are apparently NOT increased on char conversion DMAs).

SNES Cart SA-1 Character Conversion

Character Conversion Types
```
Conversion  DMA-Transfer     Source / Pixel-Format
Type 1      Automatic        BW-RAM, Packed Pixels, Bitmap Pixel Array
Type 2      Semi-Automatic   CPU, Unpacked Pixels, 8x8 Pixel Tiles
```
Both Conversion types are writing data to a temporary buffer in I-RAM:
```
I-RAM buffer 32/64/128 bytes (two 8x8 tiles at 2bit/4bit/8bit color depth)
```
From that buffer, data is forwarded to SNES (via a simultanously executed SNES DMA, ie. via ports 43xxh).

Character Conversion 1 - Automatically Convert Packed BW-RAM Pixels
Can be used only if the cartridge DOES contain BW-RAM (most or all do so).
First, do this on SA-1 side:
```
Set DCNT (Port 2230h) set to Char Conversion Type 1   (...and no DMA-enable?)
```
Then do following on SNES side:
```
Set SDA (Port 2232h-2234h)=BW-RAM offset, align by (bytes/char)*(chars/line)
Set CDMA (Port 2231h) = store bits/pixel and chars/line
Set DDA (Port 2235h-2236h)=I-RAM offset, align (bytes/char)*2 (2237h=unused)
```

```
  Wait for SFR.Bit5 (Port 2300h) Char_DMA_IRQ (=first character available)
  Launch SNES-DMA via Port 43xxh from "Virtual BW-RAM?" to PPU-VRAM
    (this can transfer the WHOLE bitmap in one pass)
```
Finally, after the SNES-DMA has finished, do this on SA-1 side:
```
  Set CDMA.Bit7=1 (Port 2231h) - terminate SA-1 DMA
    (that stops writing to I-RAM on SA-1 side)
    (and stops tile-data to be mapped to 400000h-43FFFFh on SNES-side)
```
During conversion, the SA-1 can execute other program code (but waits may occur on BW-RAM and I-RAM accesses). The SNES CPU is paused (by the DMA) for most of the time, except for the time slots shortly before/after the DMA; in that time slots, the SNES may access I-RAM, but may not access BW-RAM.
Conversion 1 is used by Haruka Naru Augusta 3 and Pebble Beach no Hotou.

Character Conversion 2 - Semi-Automatic Convert Unpacked CPU Pixels
First, do this on SA-1 side:
```
  Set DCNT (Port 2230h) set to Char Conversion Type 2 and set DMA-enable
  Set CDMA (Port 2231h) = store bits/pixel (chars/line is not used)
  Set DDA (Port 2235h-2236h)=I-RAM offset, align (bytes/char)*2 (2237h=unused)
```
Then repeat for each character:
```
  for y=0 to 7, for x=0 to 7, [2240h+x+(y and 1)]=pixel(x,y), next x,y
  On SNES side: Transfer DMA from 1st/2nd I-RAM buffer half to VRAM or WRAM
```
Finally,
```
  Set DCNT.Bit7=0 (Port 2230h) - disable DMA
```
Conversion 2 is used by Haruka Naru Augusta 3 and SD Gundam G NEXT.

SNES Cart SA-1 Arithmetic Maths

2250h SA-1 MCNT - Arithmetic Control (W)
```
  0-1 Arithmetic Mode (0=Multiply, 1=Divide, 2=MultiplySum, 3=Reserved)
  2-7 Not used (should be "..") (whatever ".." means, maybe "0"?)
```
Note: Writing Bit1=1 does reset the Sum (aka "Cumulative Sum" aka "Accumulative Sum") to zero.

2251h SA-1 MA - Arithmetic Parameter A Lsb (Multiplicand/Dividend) (W)
2252h SA-1 MA - Arithmetic Parameter A Msb (Multiplicand/Dividend) (W)
```
  0-15  SIGNED multiplicand or dividend (that is, both are signed)
```
The value in this register is kept intact after multiplaction, but gets destroyed after division.

2253h SA-1 MB - Arithmetic Parameter B Lsb (Multiplier/Divisor) (W)
2254h SA-1 MB - Arithmetic Parameter B Msb (Multiplier/Divisor)/Start (W)
```
  0-15  SIGNED multiply parameter, or UNSIGNED divisor
```
The value in this register gets destroyed after both multiplaction and division. Writing to 2254h starts the operation. Execution time is 5 cycles (in 10.74MHz units) for both Multiply and Divide, and 6 cycles for Multiply/Sum.

2306h SA-1 MR - Arithmetic Result, bit0-7 (Sum/Product/Quotient) (R)
2307h SA-1 MR - Arithmetic Result, bit8-15 (Sum/Product/Quotient) (R)
2308h SA-1 MR - Arithmetic Result, bit16-23 (Sum/Product/Remainder) (R)
2309h SA-1 MR - Arithmetic Result, bit24-31 (Sum/Product/Remainder) (R)
230Ah SA-1 MR - Arithmetic Result, bit32-39 (Sum) (R)
```
  32bit Multiply Result    (SIGNED)
  40bit Multiply/Sum       (SIGNED)
  16bit Division Result    (SIGNED)
  16bit Division Remainder (UNSIGNED !!!)
```

230Bh SA-1 OF - Arithmetic Overflow Flag (R)
This bit is reportedly set on 40bit multiply/addition overflows (rather than on more useful 32bit overflows), thereby overflow can't occur unless one is doing at least 512 continous multiply/additions.
```
  0-6 Not used (reportedly "..") (whatever ".." means, maybe 0 or open bus?)
  7   Arithmetic Sum Overflow Flag (0=No overflow, 1=Overflow)
```
Unknown when this bit gets cleared (all operations, or mode changes)?

Division by zero returns result=0000h and remainder=0000h (other info claims other values?) (but, as far as known, doesn't set set overflow flag).

SNES Cart SA-1 Variable-Length Bit Processing

2258h SA-1 VBD - Variable-Length Bit Processing (W)
```
  0-3  Data Length (1..15=1..15 bits, or 0=16 bits)
  4-6  Not used (should be "..") (whatever ".." means, maybe "0"?)
  7    Data Read Mode (0=Fixed Mode, 1=Auto-increment)
```
Manual/Fixed Mode is used by Jumpin Derby. Auto-increment isn't used by any known games.

2259h SA-1 VDA - Variable-Length Bit Game Pak ROM Start Address Lsb (W)
225Ah SA-1 VDA - Variable-Length Bit Game Pak ROM Start Address Mid (W)
225Bh SA-1 VDA - Variable-Length Bit Game Pak ROM Start Address Msb & Kick
```
  0-23  Game Pak ROM Address
```
Reading starts on writing to 225Bh.
The ROM address is probably originated at 000000h (rather than using LoROM/HiROM like CPU addresses)?

230Ch SA-1 VDP - Variable-Length Data Read Port Lsb (R)
230Dh SA-1 VDP - Variable-Length Data Read Port Msb (R)
```
  0-15  Data
```
Unknown what happens on data length less than 16bits:
```
  Are the selected bits located in MSBs or LSBs?
  Are the other bits set to zero? To next/prev values? Sign-expanded??
```
There is an "auto-increment" feature, which may trigger on reading 230Ch? or on reading or 230Dh?

```
;*******PRELOAD:
;Preload occurs after writing VDA
; bitpos = [2259h]*8
; [230Ch] = WORD[bitpos/8]
```

```
;*******INCREMENT:
;Increment occurs AFTER reading VDP (when auto-increment enabled),
;and after writing VDB (reportedly always, but SHOULD be ONLY when inc=off)?
; bitpos=bitpos+(([2258h]-1) AND 0Fh)+1
; [230Ch] = dword[bitpos/16*2] shr (bitpos and 15) AND FFFFh
```

SNES Cart GSU-n (programmable RISC CPU) (aka Super FX/Mario Chip) (10 games)

Graphic Support Unit (GSU) (10.74MHz RISC-like CPU)

GSU Opcodes

Misc

GSU Caches

Pinouts

SNES Cart GSU-n List of Games, Chips, and PCB versions

GSU1/Mario Chip1 is used by six games:

```
Dirt Racer (1994) MotiveTime/Elite Systems (EU)
Dirt Trax FX (1995) Sculptured Software/Acclaim Entertainment (NA)
Powerslide (cancelled, but unfinished prototype leaked) Elite Systems (EU)
Star Fox / Starwing (1993) Argonaut/Nintendo EAD (NA) (JP) (EU)
Star Fox / Starwing: Competition Edition (demo version) (1993) (NA) (EU)
Stunt Race FX / Wild Trax (1994) Argonaut/Nintendo EAD (NA) (JP) (EU)
Vortex (1994) Argonaut Games/Electro Brain (NA), Pack-In-Video (JP)
```

GSU2/GSU2-SP1 is used by four games:

```
Doom (1996) Sculptured Software/Williams (NA), Imagineer (JP), Ocean (EU)
Super Mario World 2: Yoshi's Island (1995) Nintendo EAD (NA) (JP) (EU)
Winter Gold / FX Skiing (1997) Funcom/Nintendo (NA) (EU)
Star Fox 2 (cancelled, but near-finished Beta version leaked into internet)
```

Reportedly, there have been another three GSU2 games planned:

```
FX Fighter (Beta) (cancelled) Argonaut Games/GTE Entertainment (NA) (EU)
Comanche (cancelled) Nova Logic (NA)
Super Mario FX (cancelled) Nintendo EAD
```

GSU Chip Versions

```
MC1      - 100pin - A/N Inc. Nintendo Mario Chip 1 (reportedly "FX-chip 1")
GSU1     - 100pin - A/N Inc. Nintendo Super FX 1 (10.74MHz RISC-like CPU)
GSU1A    - 100pin - A/N Inc. Nintendo Super FX 1
GSU2     - 112pin - A/N Inc. Nintendo Super FX 2 (as above, but 21MHz)
GSU2-SP1 - 112pin - A/N Inc. Nintendo Super FX 2 (as above, but 21MHz)
```

XXX according to MotZilla, GSU1 supports 21MHz, too? (but with less memory)

GSU PCB Versions

```
SHVC-1C0N         Mario Chip 1      Star Fox (Blob)
SHVC-1C0N5S-01    Mario Chip 1      Star Fox (SMD)
SHVC-1CA0N5S-01   GSU-1             Dirt Racer & Vortex
SHVC-1CA0N6S-01   GSU-1             Dirt Trax FX
SHVC-1CA6B-01     GSU-1 Battery     Stunt Race FX
SHVC-1CB0N7S-01   GSU-2             Doom
SHVC-1CB5B-01     GSU-2 Battery     Super Mario World 2: Yoshi's Island
SHVC-1CB5B-20     GSU-2-SP1 Battery Super Mario World 2: Yoshi's Island
SHVC-1RA2B6S-01   GSU1A Batt+Eprom  Powerslide (prototype board)
GS 0871-102       Mario Chip 1      Super Famicom Box PSS61 multi-game-cart
```

Note: Doom's "1CB0N7S" board has only 64K RAM installed (not 128K).

SNES Cart GSU-n Memory Map

MC1 Memory Map (at SNES Side)

```
00-3F/80-BF:3000-347F  GSU I/O Ports
00-1F/80-9F:8000-FFFF  Game Pak ROM in LoRom mapping (1Mbyte max)
60-7D/E0-FF:0000-FFFF  Game Pak RAM with mirrors (64Kbyte max?, usually 32K)
Other Addresses        Open Bus
```

GSU1 Memory Map (at SNES Side)

```
00-3F/80-BF:3000-34FF? GSU I/O Ports
00-3F/80-BF:6000-7FFF  Mirror of 70:0000-1FFF (ie. FIRST 8K of Game Pak RAM)
00-3F/80-BF:8000-FFFF  Game Pak ROM in LoRom mapping (1Mbyte max?)
```

```
  40-5F/C0-DF:0000-FFFF  Game Pak ROM in HiRom mapping (mirror of above)
  70-71/F0-F1:0000-FFFF  Game Pak RAM with mirrors (64Kbyte max?, usually 32K)
  78-7x/F8-Fx:0000-FFFF  Unknown (maybe Additional "Backup" RAM like GSU2)
  Other Addresses        Open Bus
```

GSU2 Memory Map (at SNES Side)

```
  00-3F/80-BF:3000-34FF  GSU I/O Ports
  00-3F/80-BF:6000-7FFF  Mirror of 70:0000-1FFF (ie. FIRST 8K of Game Pak RAM)
  00-3F:8000-FFFF        Game Pak ROM in LoRom mapping (2Mbyte max)
  40-5F:0000-FFFF        Game Pak ROM in HiRom mapping (mirror of above)
  70-71:0000-FFFF        Game Pak RAM       (128Kbyte max, usually 32K or 64K)
  78-79:0000-FFFF        Additional "Backup" RAM  (128Kbyte max, usually none)
  80-BF:8000-FFFF        Additional "CPU" ROM LoROM (2Mbyte max, usually none)
  C0-FF:0000-FFFF        Additional "CPU" ROM HiROM (4Mbyte max, usually none)
  Other Addresses        Open Bus
```

For HiROM mapping the address bits are shifted, so both LoROM and HiROM are linear (eg. Bank 40h contains mirrors of Bank 00h and 01h).
Although both LoROM and HiROM are supported, the header & exception vectors are located at ROM Offset 7Fxxh (in LoROM fashion), accordingly the cartridge header declares the cartridge as LoROM.
The additional ROM/RAM regions would be mapped to SNES CPU only (not to GSU), they aren't installed in existing cartridges, that implies that the "Fast" ROM banks (80h-FFh) are unused, so GSU games are restricted to "Slow" ROM.

GSU2 Memory Map (at GSU Side)

```
  00-3F:0000-7FFF  Mirror of LoROM at 00-3F:8000-FFFF (for "GETB R15" vectors)
  00-3F:8000-FFFF  Game Pak ROM in LoRom mapping (2Mbyte max)
  40-5F:0000-FFFF  Game Pak ROM in HiRom mapping (mirror of above 2Mbyte)
  70-71:0000-FFFF  Game Pak RAM       (128Kbyte max, usually 32K or 64K)
  PBR:0000-01FF    Code-Cache (when having manually stored opcodes in it)
```

PBR can be set to both ROM/RAM regions (or cache region), ROMBR only to ROM region (00h-5Fh), RAMBR only to RAM region (70h-71h).

GSU Interrupt Vectors
The SNES Exception Vectors (at FFE4h-FFFFh) are normally located in Game Pak ROM. When the GSU is running (with GO=1 and RON=1), ROM isn't mapped to SNES memory, instead, fixed values are appearing as ROM (depending of the lower 4bit of the address):

```
  Any Address     Exception Vectors
  [xxx0h]=0100h   -
  [xxx2h]=0100h   -
  [xxx4h]=0104h   [FFE4h]=0104h  COP Vector in 65C816 mode (COP opcode)
  [xxx6h]=0100h   [FFE6h]=0100h  BRK Vector in 65C816 mode (BRK opcode)
  [xxx8h]=0100h   [FFE8h]=0100h  ABT Vector in 65C816 mode (Not used in SNES)
  [xxxAh]=0108h   [FFEAh]=0108h  NMI Vector in 65C816 mode (Vblank)
  [xxxCh]=0100h   -
  [xxxEh]=010Ch   [FFEEh]=010Ch  IRQ Vector in 65C816 mode (H/V-IRQ & GSU-STOP)
```

It'd be best to set the Game Pak ROM vectors to the same addresses, otherwise the vectors would change when the GSU is running (or possibly, the fixed-LSBs may be mixed-up with ROM-MSBs).

GSU Cartridge Header (always at ROM Offset 7Fxxh, in LoROM fashion)

```
  [FFD5h]=20h        Set to "Slow/LoROM" (although both LoROM/HiROM works)
  [FFD6h]=13h..1Ah   Chipset = GSUn (plus battery present/absent info)
  [FFD8h]=00h        Normal SRAM Size (None) (always use the Expansion entry)
  [FFBDh]=05h..06h   Expansion RAM Size (32Kbyte and 64Kbyte exist)
  Caution: Starfox/Star Wing, Powerslide, and Starfox 2 do not have extended
  headers (and thereby no [FFBDh] entry). RAM Size for Starfox/Starwing is
  32Kbytes, RAM Size for Powerslide and Starfox 2 is unknown.
```

There is no info in the header (nor extended header) whether the game uses a GSU1 or GSU2. Games with 2MByte ROM are typically using GSU2 (though that rule doesn't always match: Star Fox 2 is only 1MByte).

GSU Busses
The GSU seems to have 4 address/data busses (three external ones, and one internal cache bus):

```
  SNES bus (for forwarding ROM/RAM access to SNES)
  ROM bus (for GSU opcode fetches, GETxx reads, and SNES reads)
  RAM bus (for GSU opcode fetches, LOAD/STORE/PLOT/RPIX, and SNES access)
  Code cache bus (for GSU opcode fetches only) (and SNES I/O via 3100h..32FFh)
```

To some level, this allows to do multiple things simultaneously: Reading a GSU opcode from cache at the same time while prefetching ROM data and forwarding the RAM or Pixel cache to RAM.

SNES Cart GSU-n I/O Map

GSU I/O Map (in banks 00h-3Fh and 80h-BFh)
During GSU operation, only SFR, SCMR, and VCR may be accessed.

```
  3000h-3001h R0  Default source/destination register (Sreg/Dreg) (R/W)
  3002h-3003h R1  PLOT opcode: X coordinate (0000h on reset) (R/W)
  3004h-3005h R2  PLOT opcode: Y coordinate (0000h on reset) (R/W)
  3006h-3007h R3  General purpose (R/W)
  3008h-3009h R4  LMULT opcode: lower 16bits of result (R/W)
  300Ah-300Bh R5  General purpose (R/W)
  300Ch-300Dh R6  LMULT and FMULT opcodes: multiplier (R/W)
  300Eh-300Fh R7  MERGE opcode (R/W)
  3010h-3011h R8  MERGE opcode (R/W)
  3012h-3013h R9  General purpose (R/W)
  3014h-3015h R10 General purpose (conventionally stack pointer) (R/W)
  3016h-3017h R11 LINK opcode: destination (R/W)
  3018h-3019h R12 LOOP opcode: counter (R/W)
  301Ah-301Bh R13 LOOP opcode: address (R/W)
  301Ch-301Dh R14 GETxx opcodes: Game Pak ROM Address Pointer (R/W)
  301Eh-301Fh R15 Program Counter, writing MSB starts GSU operation (R/W)
  3020h-302Fh -
  3030h-3031h SFR Status/Flag Register (R) (Bit1-5: R/W)
  3032h       -
  3033h       BRAMR Back-up RAM Register (W)
  3034h       PBR   Program Bank Register (8bit, bank 00h..FFh) (R/W)
  3035h       -
  3036h       ROMBR Game Pak ROM Bank Register (8bit, bank 00h..FFh) (R)
  3037h       CFGR  Config Register (W)
  3038h       SCBR  Screen Base Register (8bit, in 1Kbyte units) (W)
  3039h       CLSR  Clock Select Register (W)
  303Ah       SCMR  Screen Mode Register (W)
```

```
303Bh       VCR   Version Code Register (R)
303Ch       RAMBR Game Pak RAM Bank Register (1bit, bank 70h/71h) (R)
303Dh       -
303Eh-303Fh CBR   Cache Base Register (in upper 12bit; lower 4bit=unused) (R)
N/A         COLR  Color Register (COLOR,GETC,PLOT opcodes)
N/A         POR   Plot Option Register (CMODE opcode)
N/A         Sreg/Dreg    Memorized TO/FROM Prefix Selections
N/A         ROM Read Buffer (1 byte) (prefetched from [ROMBR:R14])
N/A         RAM Write Buffer (1 byte/word)
N/A         RAM Address (1 word, or word+rambr?) (for SBK opcode)
N/A         Pixel Write Buffer (two buffers for one 8-pixel row each)
3100h-32FFh Cache RAM
```

Full I/O Map with Mirrors for Black Blob (VCR=01h)

```
3000h..301Fh  20h  R0-R15
3020h..302Fh  10h  open bus
3030h..3031h  2    status reg
3032h..303Fh  0Eh  mirrors of status reg (except 303Bh=01h=VCR)
3040h..305Fh  20h  mirror of R0-R15
3060h..307Fh  20h  mirrors of status reg (except 307Bh=01h=VCR)
3080h..30FFh  80h  open bus
3100h..32FFh  200h cache
3300h..332Fh  30h  open bus
3330h..333Fh  10h  mirrors of status reg (except 333Bh=01h=VCR)
3340h..335Fh  20h  mirror of R0-R15
3360h..337Fh  20h  mirrors of status reg (except 337Bh=01h=VCR)
3380h..33FFh  80h  open bus
3400h..342Fh  30h  open bus
3430h..343Fh  10h  mirrors of status reg (except 343Bh=01h=VCR)
3440h..345Fh  20h  mirror of R0-R15
3460h..347Fh  20h  mirrors of status reg (except 347Bh=01h=VCR)
3480h..3FFFh  B80h open bus
```

Full I/O Map with Mirrors for GSU2 (VCR=04h)

```
3000h..301Fh  20h   R0-R15
3020h..302Fh  10h   mirror of 3030h..303Fh
3030h..303Fh  10h   status regs (unused or write-only ones return 00h)
3040h..30FFh  C0h   mirrors of 3000h..303Fh
3100h..32FFh  200h  cache
3300h..34FFh  200h  mirrors of 3000h..303Fh
3500h..3FFFh  B00h  open-bus
```

SNES Cart GSU-n General I/O Ports

3000h-301Fh - R0-R15 - CPU Registers (R/W)
16bit CPU registers (see GSU I/O map for additional details on each register).
Writes to 3000h-301Eh (even addresses) do set LATCH=data.
Writes to 3001h-301Fh (odd addresses) do apply LSB=LATCH and MSB=data.
Writes to 301Fh (R15.MSB) do also set GO=1 (and start GSU code execution).

3030h/3031h - SFR - Status/Flag Register (R) (Bit1-5: R/W)

```
0  -    Always 0                                                    (R)
1  Z    Zero Flag     (0=NotZero/NotEqual, 1=Zero/Equal)            (R/W)
2  CY   Carry Flag    (0=Borrow/NoCarry, 1=Carry/NoBorrow)          (R/W)
3  S    Sign Flag     (0=Positive, 1=Negative)                      (R/W)
4  OV   Overflow Flag (0=NoOverflow, 1=Overflow)                    (R/W)
5  GO   GSU is running (cleared on STOP) (can be forcefully=0 via 3030h)(R/W)
6  R    ROM[R14] Read (0=No, 1=Reading ROM via R14 address)         (R)
7  -    Always 0                                                    (R)
8  ALT1 Prefix Flag           ;\for ALT1,ALT2,ALT3 prefixes         (R)
9  ALT2 Prefix Flag           ;/                                    (R)
10 IL   Immediate lower 8bit flag ;\Unknown, probably set/reset internally
11 IH   Immediate upper 8bit flag ;/when processing opcodes with imm operands
12 B    Prefix Flag           ;-for WITH prefix (used by MOVE/MOVES opcodes)
13 -    Always 0                                                    (R)
14 -    Always 0                                                    (R)
15 IRQ  Interrupt Flag (reset on read, set on STOP) (also set if IRQ masked?)
```

This register is read/write-able even when the GSU is running; reading mainly makes sense for checking GO and IRQ bits, writing allows to clear the GO flag (thereby aborting the GSU program; the write does most likely also destroy the other SFR bits, so one cannot pause/resume).

3034h - PBR - Program Bank Register (8bit, bank 00h..5Fh,70h..71h) (R/W)
3036h - ROMBR - Game Pak ROM Bank Register (8bit, bank 00h..5Fh) (R)
303Ch - RAMBR - Game Pak RAM Bank Register (1bit, bank 70h..71h) (R)
Memory banks for GSU opcode/data accesses. PBR can be set to both ROM and RAM regions, ROMBR/RAMBR only to ROM or RAM regions respectively. Existing cartridges have only 32Kbyte or 64Kbyte RAM, so RAMBR should be always zero.
According to book2 (page 258), the screen base is also affected by RAMBR (unknown if that is true, theoretically, SCBR is large enough to address more than 64Kbytes without RAMBR).

303Eh/303Fh - CBR - Cache Base Register (upper 12bit; lower 4bit=unused) (R)
Code-Cache Base for Game Pak ROM/RAM. The register is read-only, so the SNES cannot directly write to it, however, the SNES can set CBR=0000h by writing GO=0 (in SFR register).

3033h - BRAMR - Back-up RAM Register (W)

```
0   BRAM Flag (0=Disable/Protect, 1=Enable)
1-7 Not used (should be zero)
```

This register would be used only if the PCB does have a separate "Backup" RAM chip mapped to 780000h-79FFFFh (additionally to the Game Pak RAM chip). None of the existing PCBs is having that extra RAM chip, so the register is having no function. (Note: However, some PCBs do include a battery wired to Game Pak RAM chip, anyways, that type of "backup" isn't affected by this register).

303Bh - VCR - Version Code Register (R)

```
0-7 GSU Chip Version (01h..0xh ?)
```

Known versions: 1=MC1/Blob, ?=MC1/SMD, ?=GSU1, ?=GSU1A, 4=GSU2, ?=GSU2-SP1.

3037h - CFGR - Config Register (W)
```
0-4 -   Not used (should be zero)
5   MS0 Multiplier Speed Select (0=Standard, 1=High Speed Mode)
6   -   Not used (should be zero)
7   IRQ Interrupt Mask (0=Trigger IRQ on STOP opcode, 1=Disable IRQ)
```
MS0 <must> be zero in 21MHz mode (ie. only CFGR.Bit5 or CLSR.Bit0 may be set).
MS0 is implemented in GSU2 (maybe also other chips), it is not implemented on Black Blob MC1 (which is always using slow multiply mode).

3039h - CLSR - Clock Select Register (W)
```
0   CLS Clock Select (0=10.7MHz, 1=21.4MHz)
1-7 -   Not used (should be zero)
```
CLS exists on all GSU variants (including Black Blob MC1) (however, there are rumours that the fast mode was "bugged" on older MC1, unknown if that's true).

N/A - ROM Buffer - Prefetched Byte(s?) at [ROMBR:R14]
N/A - Sreg/Dreg - Memorized TO/FROM Prefix Selections
3100h..32FFh - Cache RAM

SNES Cart GSU-n Bitmap I/O Ports

3038h - SCBR - Screen Base Register (8bit, in 1Kbyte units) (W)
```
0-7  Screen Base in 1K-byte Units (Base = 700000h+N*400h)
```

303Ah - SCMR - Screen Mode Register (W)
```
0-1 MD0-1 Color Gradient (0=4-Color, 1=16-Color, 2=Reserved, 3=256-Color)
2   HT0   Screen Height  (0=128-Pixel, 1=160-Pixel, 2=192-Pixel, 3=OBJ-Mode)
3   RAN   Game Pak RAM bus access (0=SNES, 1=GSU)
4   RON   Game Pak ROM bus access (0=SNES, 1=GSU)
5   HT1   Screen Height  (MSB of HT0 bit)
6-7 -     Not used (should be zero)
```
RON/RAN can be temporarily cleared during GSU operation, this causes the GSU to enter WAIT status (if it accesses ROM or RAM), and continues when RON/RAN are changed back to 1.
Note that "OBJ Mode" can be also selected by POR.Bit4 (if so, HT0/HT1 bits are ignored).

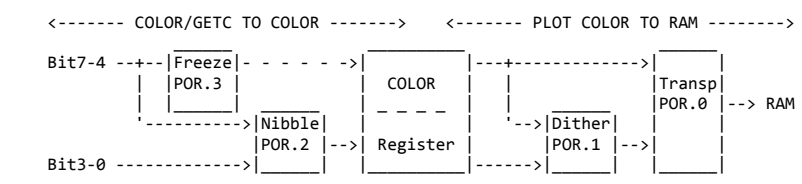
```
256x128 pixels   256x160 pixels   256x192 pixels   OBJ Mode 256x256 pixel
000 010 .. 1F0 | 000 014 .. 26C | 000 018 .. 1E8 | 000 .. 00F 100 .. 10F
001 011 .. 1F1 | 001 015 .. 26D | 001 019 .. 1E9 | ..  .. ..  ..  .. ..
..  ..  .. ..  | ..  ..  .. ..  | ..  ..  .. ..  | 0F0 .. 0FF 1F0 .. 1FF
..  ..  .. ..  | ..  ..  .. ..  | ..  ..  .. ..  | 200 .. 20F 300 .. 30F
00E 01E .. 1FE | 012 026 .. 27E | 016 02E .. 2FE | ..  .. ..  ..  .. ..
00F 01F .. 1FF | 013 027 .. 27F | 017 02F .. 2FF | 2F0 .. 2FF 3F0 .. 3FF
```
In the first three cases, BG Map is simply filled with columns containing increasing tile numbers. The fourth case is matched to the SNES two-dimensional OBJ mapping; it can be used for BG Map (with entries 0..3FF as shown above), or for OBJ tiles (whereas, mind that the SNES supports only 0..1FF OBJs, not 200..3FF).

The Tile Number is calculated as:
```
Height 128 --> (X/8)*10h + (Y/8)
Height 160 --> (X/8)*14h + (Y/8)
Height 192 --> (X/8)*18h + (Y/8)
OBJ Mode --> (Y/80h)*200h + (X/80h)*100h + (Y/8 AND 0Fh)*10h + (X/8 AND 0Fh)
```
The Tile-Row Address is:
```
4 Color Mode    TileNo*10h + SCBR*400h + (Y AND 7)*2
16 Color Mode   TileNo*20h + SCBR*400h + (Y AND 7)*2
256 Color Mode  TileNo*40h + SCBR*400h + (Y AND 7)*2
```
With Plane0,1 stored at Addr+0, Plane 2,3 at Addr+10h, Plane 4,5 at Addr+20h, Plane 6,7 at Addr+30h.

N/A - COLR - Color Register
```
0-7 CD0-7 Color Data
```

N/A - POR - Plot Option Register (CMODE)
```
0   PLOT Transparent       (0=Do Not Plot Color 0, 1=Plot Color 0)
1   PLOT Dither            (0=Normal, 1=Dither; 4/16-color mode only)
2   COLOR/GETC High-Nibble (0=Normal, 1=Replace incoming LSB by incoming MSB)
3   COLOR/GETC Freeze-High (0=Normal, 1=Write-protect COLOR.MSB)
4   OBJ Mode               (0=Normal, 1=Force OBJ mode; ignore SCMR.HT0/HT1)
5-7 Not used (should be zero)
```
Can be changed by CMODE opcode, used for COLOR/GETC/PLOT opcodes.
Dither can mix transparent & non-transparent pixels.

Bit0=0 (Transparent) causes PLOT to skip color 0 (so PLOT does only increment R1 (X-coordinate), but doesn't draw a pixel). Depending on color depth, the color 0 check tests the lower 2/4/8 bits of the drawing color (if POR.Bit3 (Freeze-High) is set, then it checks only the lower 2/4 bits, and ignores upper 4bit even when in 256-color mode).
Bit1=1 (Dither) causes PLOT to use dithering, that is, if "(r1.bit0 XOR r2.bit0)=1" then COLOR/10h is used as drawing color; using Color 0 as one of the two colors can produce a semi-transparency effect. Dither is ignored in 256-color mode.
Bit2=1 (High-Nibble) causes COLOR/GETC to replace the LSB of the incoming data by the MSB of the incoming data; this allows two 4bit bitmaps being stored at the same memory area (one in the LSBs, the other in MSBs).
Bit3=1 (Freeze-High) causes COLOR/GETC to change only the LSB of the color register; this allows the MSB to be used as fixed palette-like value in 256-color mode, it might be also useful for fixed dither-colors in 4/16 color mode.
Bit3=1 forces OBJ Mode (same as when setting SCMR.HT0/HT1 to OBJ Mode).



SNES Cart GSU-n CPU MOV Opcodes

GSU MOV Opcodes (Register/Immediate)
```
  Opcode     Clks Flags   Native       Nocash
  2s 1d         2 000---- MOVE Rd,Rs   mov Rd,Rs   ;Rd=Rs
  2d Bs         2 000vs-z MOVES Rd,Rs  movs Rd,Rs  ;Rd=Rs (with flags, OV=bit7)
  An pp         2 000---- IBT Rn,#pp   mov Rn,pp   ;Rn=SignExpanded(pp)
  Fn xx yy      3 000---- IWT Rn,#yyxx mov Rn,yyxx ;Rn=yyxx
```

GSU MOV Opcodes (Load BYTE from ROM)
```
  EF          1-6 000---- GETB         movb Rd,[romb:r14]    ;hi=zero-expanded
  3D EF       2-6 000---- GETBH        movb Rd.hi,[romb:r14] ;lo=unchanged
  3E EF       2-6 000---- GETBL        movb Rd.lo,[romb:r14] ;hi=unchanged
  3F EF       2-6 000---- GETBS        movbs Rd,[romb:r14]   ;hi=sign-expanded
```

GSU MOV Opcodes (Load/Store Byte/Word to/from RAM)
```
  3D 4n         6 000---- LDB (Rn)     movb Rd,[ramb:Rn]  ;Rd=Byte[..] ;n=0..11
  4n            7 000---- LDW (Rn)     mov Rd,[ramb:Rn]   ;Rd=Word[..] ;n=0..11
  3D Fn lo hi  11 000---- LM Rn,(hilo) mov Rn,[ramb:hilo] ;Rn=Word[..]
  3D An kk     10 000---- LMS Rn,(yy)  mov Rn,[ramb:kk*2] ;Rn=Word[..]
  3D 3n       2-5 000---- STB (Rn)     movb [ramb:Rn],Rs  ;Byte[..]=Rs ;n=0..11
  3n          1-6 000---- STW (Rn)     mov [ramb:Rn],Rs   ;Word[..]=Rs ;n=0..11
  3E Fn lo hi 4-9 000---- SM (hilo),Rn mov [ramb:hilo],Rn ;Word[..]=Rn
  3E An kk    3-8 000---- SMS (yy),Rn  mov [ramb:kk*2],Rn ;Word[..]=Rn
  90          1-6 000---- SBK          mov [ram:bk],Rs    ;Word[LastRamAddr]=Rs
```
Words at odd addresses are accessing [addr AND NOT 1], with data LSB/MSB swapped. LDB does zero-expand result (Rd.hi=00h). STB does store Rs.lo (ignores Rs.hi). SBK does "writeback" to most recently used RAM address (eg. can be used after LM) (unknown if whole 17bit, including ramb, are saved).

GSU ROM/RAM Banks
```
  3E DF         2 000---- RAMB         movb ramb,Rs ;RAMBR=Rs & 01h ;RAM Bank
  3F DF         2 000---- ROMB         movb romb,Rs ;ROMBR=Rs & FFh ;ROM Bank
```

GSU Bitmap Opcodes
```
  3D 4E         2 000---- CMODE        movb por,Rs           ;=Rs&1Fh
  4E            1 000---- COLOR        movb color,Rs         ;=Rs&FFh
  DF          1-6 000---- GETC         movb color,[romb:r14] ;=[membyte]
  4C         1-48 000---- PLOT         plot [r1,r2],color ;Pixel=COLR, R1=R1+1
  3D 4C     20-74 000-s-z RPIX         rpix Rd,[r1,r2] ;Rd=Pixel? FlushPixCache
```
Unknown if RPIX always sets SF=0, theoretically 2bit/4bit/8bit pixel-colors cannot be negative, unless it uses bit7 as sign, or so?

SNES Cart GSU-n CPU ALU Opcodes

GSU ALU Opcodes
```
  Opcode     Clks Flags   Native       Nocash
  5n            1 000vscz ADD Rn       add Rd,Rs,Rn ;Rd=Rs+Rn
  3E 5n         2 000vscz ADD #n       add Rd,Rs,n  ;Rd=Rs+n
  3D 5n         2 000vscz ADC Rn       adc Rd,Rs,Rn ;Rd=Rs+Rn+Cy
  3F 5n         2 000vscz ADC #n       adc Rd,Rs,n  ;Rd=Rs+n+Cy
  6n            1 000vscz SUB Rn       sub Rd,Rs,Rn ;Rd=Rs-Rn
  3E 6n         2 000vscz SUB #n       sub Rd,Rs,n  ;Rd=Rs-n
  3D 6n         2 000vscz SBC Rn       sbc Rd,Rs,Rn ;Rd=Rs-Rn-(Cy XOR 1)
  3F 6n         2 000vscz CMP Rn       cmp Rs,Rn    ;Rs-Rn
  7n            1 000-s-z AND Rn       and Rd,Rs,Rn ;Rd=Rs AND Rn     ;n=1..15!
  3E 7n         2 000-s-z AND #n       and Rd,Rs,n  ;Rd=Rs AND n      ;n=1..15!
  3D 7n         2 000-s-z BIC Rn       bic Rd,Rs,Rn ;Rd=Rs AND NOT Rn ;n=1..15!
  3F 7n         2 000-s-z BIC #n       bic Rd,Rs,n  ;Rd=Rs AND NOT n  ;n=1..15!
  Cn            1 000-s-z OR  Rn       or  Rd,Rs,Rn ;Rd=Rs OR Rn      ;n=1..15!
  3E Cn         2 000-s-z OR  #n       or  Rd,Rs,n  ;Rd=Rs OR n       ;n=1..15!
  3D Cn (?)     2 000-s-z XOR Rn       xor Rd,Rs,Rn ;Rd=Rs XOR Rn     ;n=1..15?
  3F Cn (?)     2 000-s-z XOR #n       xor Rd,Rs,n  ;Rd=Rs XOR n      ;n=1..15?
  4F            1 000-s-z NOT          not Rd,Rs    ;Rd=Rs XOR FFFFh
```

GSU Rotate/Shift/Inc/Dec Opcodes
```
  03            1 000-0cz LSR          shr Rd,Rs,1  ;Rd=Rs SHR 1
  96            1 000-scz ASR          sar Rd,Rs,1  ;Rd=Rs SAR 1
  04            1 000-scz ROL          rcl Rd,Rs,1  ;Rd=Rs RCL 1 ;\through
  97            1 000-scz ROR          rcr Rd,Rs,1  ;Rd=Rs RCR 1 ;/carry
  3D 96         2 000-scz DIV2         div2 Rd,Rs   ;Rd=Rs SAR 1, Rd=0 if Rs=-1
  Dn            1 000-s-z INC Rn       inc Rn       ;Rn=Rn+1          ;n=0..14!
  En            1 000-s-z DEC Rn       dec Rn       ;Rn=Rn-1          ;n=0..14!
```

GSU Byte Operations
```
  4D            1 000-s-z SWAP         ror Rd,Rs,8    ;Rd=Rs ROR 8
  95            1 000-s-z SEX          movbs Rd,Rs    ;Rd=SignExpanded(Rs&FFh)
  9E            1 000-s-z LOB          and Rd,Rs,0FFh ;Rd=Rs AND FFh  ;SF=Bit7
  C0            1 000-s-z HIB          shr Rd,Rs,8    ;Rd=Rs SHR 8    ;SF=Bit7
  70            1 000xxxx MERGE        merge Rd,r7,r8 ;Rd=R7&FF00 + R8/100h
```
Flags for MERGE are:
```
  S = set if (result AND 8080h) is nonzero
  V = set if (result AND C0C0h) is nonzero
  C = set if (result AND E0E0h) is nonzero
  Z = set if (result AND F0F0h) is nonzero (not set when zero!)
```

GSU Multiply Opcodes
```
  9F          4,8 000-scz FMULT     smulw Rd:nul,Rs,r6 ;Rd=signed(Rs*R6/10000h)
  3D 9F       5,9 000-scz LMULT     smulw Rd:R4,Rs,R6  ;Rd:R4=signed(Rs*R6)
  8n          1,2 000-s-z MULT Rn       smulb Rd,Rs,Rn ;Rd=signed(RsLsb*RnLsb)
  3E 8n       2,3 000-s-z MULT #n       smulb Rd,Rs,n  ;Rd=signed(RsLsb*0..15)
  3D 8n       2,3 000-s-z UMULT Rn      umulb Rd,Rs,Rn ;Rd=unsigned(RsLsb*RnLsb)
  3F 8n (?)   2,3 000-s-z UMULT #n      umulb Rd,Rs,n  ;Rd=unsigned(RsLsb*0..15)
```
The multiply speed can be selected via CFGR register. Do not use FMULT with Dreg=R4 (this will reportedly leave R4 unchanged). When using LMULT with Dreg=R4 then the result will be R4=MSB (and LSB is lost). Ie. if that is true then, strangely, LMULT R4 <does> work as how FMULT R4 <should> work.

SNES Cart GSU-n CPU JMP and Prefix Opcodes

GSU Special Opcodes

```
  Opcode     Clks Flags   Native    Nocash
  00            1 000---- STOP      stop  ;SFR.GO=0, SFR.IRQ=1, R15=$+2
  01            1 000---- NOP       nop   ;NOP (often used as dummy after jump)
  02           1* 000---- CACHE     cache ;IF CBR<>PC&FFF0 then CBR=PC&FFF0
```

STOP at $+0 does prefetch another opcode byte at $+1 (but without executing it), and does then stop with R15=$+2, SFR.GO=0, SFR.IRQ=1 (that, even if IRQ is disabled in CFGR.IRQ).
BUG: On MC1 (maybe also GSU1), STOP hangs when executed after a RAM write (there must be at least 2 cycles after write, eg. insert two NOPs before STOP; the required delay might vary depending on CPU speed or code cache? the bug doesn't occur on GSU2).

GSU Jump Opcodes

```
  Opcode     Clks Flags   Native    Nocash
  05 nn         2 ------- BRA addr  jr  addr   ;Always, R15=R15+signed(nn)
  06 nn         2 ------- BGE addr  jge addr   ;If (S XOR V)=0 then ..
  07 nn         2 ------- BLT addr  jl  addr   ;If (S XOR V)=1 then ..
  08 nn         2 ------- BNE addr  jne addr   ;If ZF=0 then R15=R15+signed(nn)
  09 nn         2 ------- BEQ addr  je  addr   ;If ZF=1 then R15=R15+signed(nn)
  0A nn         2 ------- BPL addr  jns addr   ;If SF=0 then R15=R15+signed(nn)
  0B nn         2 ------- BMI addr  js  addr   ;If SF=1 then R15=R15+signed(nn)
  0C nn         2 ------- BCC addr  jnc addr   ;If CY=0 then R15=R15+signed(nn)
  0D nn         2 ------- BCS addr  jc  addr   ;If CY=1 then R15=R15+signed(nn)
  0E nn         2 ------- BVC addr  jno addr   ;If OV=0 then R15=R15+signed(nn)
  0F nn         2 ------- BVS addr  jo  addr   ;If OV=1 then R15=R15+signed(nn)
  9n            1 000---- JMP Rn    jmp Rn     ;R15=Rn                ;n=8..13!
  3D 9n         2 000---- LJMP Rn   jmp Rn:Rs  ;R15=Rs, PBR=Rn, CBR=? ;n=8..13!
  3C            1 000-s-z LOOP    loop r12,r13 ;r12=r12-1, if Zf=0 then R15=R13
  9n            1 000---- LINK #n link r11,addr;R11=R15+n             ;n=1..4
```

Jumps can be also implemented by using R15 (PC) as destination register (eg. in MOV/ALU commands).
Observe that the NEXT BYTE after any jump/branch opcodes is fetched before continuing at the jump destination address. The fetched byte is executed after the jump, but before executing following opcodes at the destination (in case of multi-byte opcodes, this results in a 1-byte-fragment being located after the jump-origin, and the remaining byte(s) at the destination).

GSU Prefix Opcodes

ALT1/ALT2/ALT3 prefixes do change the operation of an opcode, these are usually implied in the opcode description (for example, "3F 6n" is "CMP R0,Rn").
TO/WITH/FROM prefixes allow to select source/destination registers (otherwise R0 is used as default register) (for example, "Bs 3F 6n" is "CMP Rs,Rn").
The prefixes are normally reset after execution of any opcode, the only exception are the Bxx (branch) opcodes, these leave prefixes unchanged (allowing to "split" opcodes, for example placing ALT1/TO/etc. before Bxx, and the next opcode byte after Bxx).
Aside from setting Sreg+Dreg, WITH does additionally set the B-flag, this causes any following 1nh/Bnh bytes to act as MOVE/MOVES opcodes (rather than as TO/FROM prefixes).

```
  Opcode Clks Flags   Name    Bflg ALT1 ALT2 Rs Rd
  3D        1 -1----- ALT1     -    1    -   -  -  ;prefix for 3D xx opcodes
  3E        1 --1---- ALT2     -    -    1   -  -  ;prefix for 3E xx opcodes
  3F        1 -11---- ALT3     -    1    1   -  -  ;prefix for 3F xx opcodes
  1n        1 ------- TO Rn    -    -    -   -  Rn ;select Rn as Rd
  2n        1 1------ WITH Rn  1    -    -   Rn Rn ;select Rn as Rd & Rs
  Bn        1 ------- FROM Rn  -    -    -   Rn -  ;select Rn as Rs
  05..0F nn 2 ------- Bxx addr -    -    -   -  -  ;branch opcodes (no change)
  other    .. 000---- other    0    0    0   R0 R0 ;other opcodes (reset all)
```

Other opcodes do reset B=0, ALT1=0, ALT2=0, Sreg=R0, Dreg=R0; that does really apply to ALL other opcodes, namely including JMP/LOOP (unlike Bxx branches), NOP (ie. NOP isn't exactly <no> operation), MOVE/MOVES (where 1n/Bn are treated as 'real' opcodes rather than as TO/FROM prefixes).

Ignored Prefixes

ALT1/ALT2 prefixes are ignored if the opcode doesn't exist (eg. if "3D xx" doesn't exist, then the CPU does instead execute "xx") (normally, doing that wouldn't make any sense, however, "Doom" is using ALT1/ALT2 alongside with conditional jumps, resulting in situations where the prefix is used/ignored depending on the jump condition).
ALT3 does reportedly mirror to ALT1 (eg. if "3F xx" doesn't exist, then it acts as "3D xx", and, if that doesn't either, as "xx").
TO/WITH/FROM are ignored if the following opcode doesn't use Dreg/Sreg.

Program Counter (R15) Notes

R15 can be used as source operand in MOV/ALU opcodes (and is also implied as such in Bxx,LINK,CACHE opcodes); in all cases R15 contains the address of the next opcode.

SNES Cart GSU-n CPU Pseudo Opcodes

Official GSU Pseudo/Macro Opcodes

```
  --            3 000---- LEA Rn,yyxx    ;Alias for IWT, without "#"
  --            - 000---- MOVE Rn,#hilo  ;Alias for IBT/IWT (depending on size)
  --            - 000---- MOVE Rn,(xx)   ;Alias for LM/LMS (depending on size)
  --            - 000---- MOVE (xx),Rn   ;Alias for SM/SMS (depending on size)
  --            - 000---- MOVEB Rn,(Rm)  ;Alias for LDB/TO+LDB (depending Rn)
  --            - 000---- MOVEB (Rm),Rn  ;Alias for STB/FROM+STB
  --            - 000---- MOVEW Rn,(Rm)  ;Alias for LDW/TO+LDW (depending Rn)
  --            - 000---- MOVEW (Rm),Rn  ;Alias for STW/FROM+STW
```

Above are official pseudo opcodes for native syntax (the nocash syntax "MOV" opcode is doing that things by default).

Nocash GSU Pseudo Opcodes

```
   jmp  nnnn      alias for "mov r15,nnnn"
   jz/jnz/jae/jb  alias for "je/jne/jc/jnc"
```

Further possible pseudo opcodes (not yet supported in a22i):

```
   push rs        mov [r10],rs, 2xinc_r10     ;\INCREASING on PUSH? or MEMFILL?
   pop  rd        2xdec_r10, mov rd,[r10]     ;/ (see Star Fox 1:ACA4)
   cmp  rn,0      alias for "sub rn,rn,0"
   call           alias for link+jmp
   ret            alias for jmp r11
   alu  rd,op     short for "alu rd,rs,op"
   and  rd,rs,n   alias for "bic rd,rs,not n"
```

SNES Cart GSU-n CPU Misc

Uncached ROM/RAM-Read-Timings

```
ROM Read:   5 cycles per byte at 21MHz, or 3 cycles per byte at 10MHz
RAM Write: 10 cycles per word at 21MHz, or unknown at 10MHz?
RAM Write: unknown number of cycles per byte?
ROM/RAM Opcode-byte-read: 3 cycles at both 21MHz and 10MHz?
```

The uncached timings aren't well documented. Possibly ROM/RAM-byte read/write are all having the same timing (3/5 clks at 10/21MHz) (and RAM-word 6/10)?

Jump Notes

Jumps can be implemented by JMP/Bxx opcodes, or by using R15 as destination register. In all cases, the next BYTE after the jump opcode is fetched as opcode byte, and is executed before continuing at the jump-target address. Possible situations are:

```
1) jump + NOP                ;very simple
2) jump + ONE-BYTE-OPCODE    ;still quite simple
3) jump + MULTI-BYTE-OPCODE  ;rather strange
4) Prefix + jump + ONE-BYTE-SUFFIX
5) Prefix + jump + MULTI-BYTE-SUFFIX
```

In case 3, the first opcode-byte is picked from the address after jump, the following byte(s) from the jump-destination.
In case 4/5, the prefix is located before the jump, the next byte after the jump (this works only with Bxx jumps) (whilst JMP/LJMP or MOV/ALU R15,dest do reset the prefix), and any further bytes at the jump-destination.

Mistakes in book2.pdf

BGE/BLT are exchanged with each other. MOVES src/dst operands are exchanged. LJMP bank/offs operands are exchanged.

GSU Undoc opcodes

UMULT #n, WITH, XOR Rn, XOR #n are sorts of undocumented; they should be described (on page 280), but the alphabetical list ends abruptly after UMULT Rn. However, they are listed in the summary (page 101) and in the index (page 409). The WITH opcode is also mentioned in various other places.

page 121: R15 after STOP (strange, is that true?) (yes, it is)
page 122: cache/cbr after ABORT

MOV R13,R15 sets R13 to addr of next opcode after MOV (eg. for LOOP start)
LINK n sets R11 to addr+n of next opcode (eg. for "CALLs" via jmp)

GSU Power Consumption

The GSU does (when it is running) increase the power consumption, this can overload the SNES power supply if additional peripherals are connected. GSU software should detect which controllers are connected, and refuse to start the GSU if a controller with high power consumption (or with unknown power consumption) is connected. The standard joypads are okay. A Multiplayer 5 adaptor isn't okay (at least, when multiple controllers are connected to it).

After STOP

Restarting (somewhere(?) after STOP) is possible by setting GO-flag (done by Dirt Trax FX).

SNES Cart GSU-n Code-Cache

ROM/RAM-Code-Cache (512-byte cache)

This cache is used only for Opcode fetches from ROM or RAM (not for reading/writing Data to/from ROM nor RAM) (however, it does slightly increase data access speed in so far that data can be read/written via Gamepak bus, simultaneously while fetching opcodes from the cache).
32 lines of 16-bytes.
CACHE
LJMP
STOP
ABORT
after STOP, one "must" clear GO by software to clear the cache

Cache Area

"SNES_Addr = (CBR AND 1FFh)+3100h". For example, a CACHE opcode at C3A5h will set CBR to C3A0h, and the (initially empty) cached region will be C3A0h..C59Fh, when code gets loaded into the cache, GSU:C3A0h..C3FFh shows up at SNES:32A0h..32FFh, and GSU:C400h..C59Fh at SNES:3100h..329Fh.

Writing to Code-Cache (by SNES CPU)

First of, set GO=0 (ie. write SFR=0000h), this forces CBR=0000h, and marks all cache lines as empty. Then write opcodes 16-byte lines at 3100h..32FFh, writing the last byte of a line at [3xxFh] will mark the line as not-empty.
Thereafter, the cached code can be excuted (by setting R15 to 0000h..01Fxh), in this case, the GSU can be operated without RON/RAN flags being set - unless R15 leaves the cached area (this occurs also when a STOP is located in last byte of last cache line; the hardware tries to prefetch one byte after STOP), or unless ROM-DATA is accessed (via GETxx opcodes) or unless RAM-DATA is accessed (via LOAD/STORE or PLOT/RPIX opcodes). Ie. usually one would have RAN set (unless all incoming/outgoing parameters can be passed though R0..R14 registers).

Code-Cache Loading Notes

The 16-byte cache-lines are loaded alongside while executing opcodes (rather than first loading the whole 16-byte-line, and then executing the opcodes within it; which would be slightly slower). There are two special cases related to jumps: If current cache-line isn't fully loaded then hardware keeps loading the remaining bytes (from jump-origin to end-of-line). If the jump-target isn't aligned by 16 (and isn't yet cached), then the hardware loads the leading bytes (from start-of-line to jump-target). After that two steps, normal execution continues at the jump-target address.
The leading-stuff also occurs on CACHE instruction.

```
CACHE sets CBR to "R15 AND FFF0h" (whereas R15=address after CACHE opcode)
LJMP sets CBR to "R15 AND FFF0h" (whereas R15=jump target address)
SNES write to SFR register with GO=0 sets CBR=0000h
(all of the above three cases do also mark all cache lines as empty)
```

All Code-Cache lines are marked as empty when executing CACHE or LJMP opcodes, or when the SNES clears the GO flag (by writing to SFR). The STOP opcode however (which also clears GO), doesn't empty the cache, so one may eventually re-use the cached values when restarting the GSU (however, if PBR or code in GamePak RAM has changed, then one must clear the cache by writing GO=0).

According to cache description (page 132), Cache-Code is 6 times faster than ROM/RAM. However, according to opcode descriptions (page 160 and up), cache is only 3 times faster than ROM/RAM. Whereas, maybe 6 times refers to 21MHz mode, and 3 times to 10MHz mode?

The CACHE opcode is typically executed prior to loops and/or at the begin of often-used sub-functions (or ideally, the loop and any used subfunctions should both fit into the 512-byte cache region).

SNES Cart GSU-n Pixel-Cache

RAM-Pixel-Write-Cache (two 8-pixel rows)

pixel cache is flushed when:
```
1) cache full
2) doing rpix <--- this does also WAIT until it is flushed
3) changing r1 or r2 (really?)
```

Pixel Cache
Primary Pixel Cache (written to by PLOT)
Secondary Pixel Cache (data copied from Primary Cache, this WAITs if Secondary cache wasn't yet forwarded to RAM) (if less than 8 flags are set, data is merged with old RAM data).
Each cache contains 8 pixels (with 2bit/4bit/8bit depth), plus 8 flags (indicating if (nontransparent) pixels were plotted).
Pixel X/Y coordinates are 8bit wide (using LSBs of R1/R2 registers).
(X and F8h) and (Y and FFh) are memorized, when plotting to different values, Primary cache is forwarded to Secondary Cache, this happens also when all 8 cache flags are set.
Do not change SCREEN MODE (how to do that at all while GSU is running? SCMR is writeable for changing RAN/RON, but changing the other SCMR bits during GSU execution would be rather unpredictable) when data is in pixel caches (use RPIX to force the caches to be flushed). Before STOP opcode, do also use RPIX to force the caches to be flushed.
RPIX isn't cached, it does always read data from RAM, not from cache. Moreover, before reading RAM, RPIX does force both pixel caches (unless they are empty) to be forwarded to RAM. This is making RPIX very slow (trying to read/modify pixels via RPIX+PLOT would work very slow). So far, RPIX is mainly useful for forcing the pixel caches to be forwarded to RAM (and to WAIT until that forwarding has completed).

SNES Cart GSU-n Other Caches

ROM-Read-Data Cache (1-byte read-ahead)
The cache is used for GETB/GETBS/GETBL/GETBH/GETC opcodes (which do read from [ROMBR:R14]). Loading the cache is invoked by any opcodes that do change R14 (such like ADD,MOVE,etc.), allowing following GETxx opcodes to be executed without Waitstates.
In some situations WAITs can occur: When the cache-load hasn't yet completed (ie. GETxx executed shortly after changing R14), when an opcode is fetched from ROM (rather than from RAM or Code-Cache), when ROMBR is changed (caution: in this special case following GETxx will receive [OldROMBR:R14] rather than [NewROMBR:R14]).
Caution: Do not execute the CACHE opcode shortly (7 cycles in 21MHz mode, or 4 cycles in 10MHz mode) after changing R14 (when doing that, the read from R14 will fail somehow, and following GETxx will return garbage).
Unknown if SNES writes to R14 (via Port 301Ch) do also prefetch [R14] data?

RAM-Write-Data Cache (1-byte/1-word write queue)
This cache is used for STB/STW/SM/SMS/SBK opcodes. After any such store opcodes, the written byte/word is memorized in the cache, and further opcodes can be fetched (from ROM or from Code-Cache) immediately without Waitstates, simultaneously with the cached value being forwarded to RAM.
In some situations WAITs can occur: When cache already contained data (ie. when executing two store opcodes shortly after each other), when an opcode is fetched from RAM (rather than from ROM or Code-Cache), when the RAMBR register is changed (this works as expected, it finishes the write to [OldRAMBR:nnnn]).
Results on doing Data-RAM-Reads while the Data-RAM-write is still busy are unknown (possibly, this will WAIT, too) (or it may return garbage)?
WAITs should also occur when the pixel-cache gets emptied?

RAM-Address-Cache (1 word) (Bulk Processing for read-modify-write)
This very simple cache memorizes the most recently used RAM address (from LM/LMS opcodes, and probably also from LDB/LDW/STB/STW/SM/SMS opcodes; though some games insert STW to push data on stack, as if they were intended not to change the memorized address?), the SBK opcode can be used to write a word to the memorized address (ie. one can avoid repeating immediate operands in SM/SMS opcodes).

SNES Cart Capcom CX4 (programmable RISC CPU) (Mega Man X 2-3) (2 games)

SNES Cart Capcom CX4 - I/O Ports
SNES Cart Capcom CX4 - Opcodes
SNES Cart Capcom CX4 - Functions
SNES Pinouts CX4 Chip

Capcom CX4 - 80pin chip
Used only by two games:
```
Mega Man X2 (1994) Capcom (NA) (JP) (EU)   ;aka Rockman X2
Mega Man X3 (1995) Capcom (NA) (JP)
```
The CX4 chip is actually a Hitachi HG51B169 as confirmed by decapping.
Note: The CX4 is occassionally referred to as C4 (the real chip name is CX4, the C4 variant is some kind of scene slang).

CX4 Memory Map
```
I/O  00-3F,80-BF:6000-7FFF
ROM  00-3F,80-BF:8000-FFFF
SRAM 70-77:0000-7FFF (not installed; reads return 00h)
```

MISC MISC MISC
Commands are executed on the CX4 by writing the command to 0x7F4F while bit 6 of 0x7F5E is clear. Bit 6 of 0x7F5E will stay set until the command has completed, at which time output data will be available.
[Registers]
```
$7f49-b = ROM Offset
$7f4d-e = Page Select
$7f4f = Instruction Pointer
Start Address = ((Page_Select * 256) + Instruction Pointer) * 2) + ROM_Offset
```
[Memory layout]
```
Program ROM is obviously 256x16-bit pages at a time. (taken from the SNES ROM)
Program RAM is 2x256x16-bit. (two banks)    ;<-- uh, that means cache?
Data ROM is 1024x24-bit. (only ROM internal to the Cx4)
Data RAM is 4x384x16-bit.                   ;<-- uh, but it HAS 8bit data bus?
Call stack is 8-levels deep, at least 16-bits wide.
```

CX4ROM (3Kbytes) (1024 values of 24bit each)

```
Index      Name  ;Entry     = Table Contents   = Formula
-------------------------------------------------------------------------
000..0FFh  Div   ;N[0..FFh] = FFFFFFh..008080h = 800000h/(00h..FFh)
100..1FFh  Sqrt  ;N[0..FFh] = 000000h..FF7FDFh = 100000h*Sqrt(00h..FFh)
200..27Fh  Sin   ;N[0..7Fh] = 000000h..FFFB10h = 1000000h*Sin(0..89')
280..2FFh  Asin  ;N[0..7Fh] = 000000h..75CEB4h = 800000h/90'*Asin(0..0.99)
300..37Fh  Tan   ;N[0..7Fh] = 000000h..517BB5h = 10000h*Tan(0..89')
380..3FFh  Cos   ;N[0..7Fh] = FFFFFFh..03243Ah = 1000000h*Cos(0..89')
```

Sin/Asin/Tan/Cos are spanning only 90' out of 360' degress (aka 80h out of 200h degrees). Overflows on Div(0) and Cos(0) are truncated to FFFFFFh. All values are unsigned, and all (except Asin/Tan) are using full 24bits (use SHR opcode to convert these to signed values with 1bit sign + 23bit integer; for Div one can omit the SHR if divider>01h).

CX4 Component List (Megaman X2)

```
PCB "SHVC-2DC0N-01, (C)1994 Nintendo"
U1 32pin P0 8M MASK ROM  (LH538LN4 = 8Mbit)
U2 32pin P1 4/8 MASK ROM (LH534BN2 or LH5348N2 or so = 4Mbit)
U3 80pin CX4 (CAPCOM CX4 DL-2427, BS169FB)
U4 18pin CIC (F411A)
X1  2pin 20MHz
J  62pin Cart Edge connector (unknown if any special pins are actually used)
```

CX4 Component List (Megaman X3)

```
PCB "SHVC-1DC0N-01, (C)1994 Nintendo"
U1 40pin MASK ROM  (TC5316003CF = 16Mbit)
U2 80pin CX4 (CAPCOM CX4 DL-2427, BS169FB)
U3 18pin CIC (F411A)
X1  2pin 20MHz
J  62pin Cart Edge connector (unknown if any special pins are actually used)
```

CX4 Cartridge Header (as found in Mega Man X2/X3 games)

```
[FFBD]=00h ;expansion RAM size (none) (there is 3KB cx4ram though)
[FFBF]=10h ;CustomChip=CX4
[FFD5]=20h ;Slow LoROM (but CX4 opcodes are probably using a faster cache)
[FFD6]=F3h ;ROM+CustomChip (no battery, no sram)
[FFD7]=0Bh ;rom size (X2: 1.5MB, rounded-up to 2MB) (X3: real 2MB)
[FFD8]=00h ;sram size (none) (there is 3KB cx4ram though)
[FFDA]=33h ;Extended Header (with FFB0h-FFBFh)
```

ROM Enable

On SHVC-2DC0N-01 PCBs (ie. PCBs with two ROM chips), the 2nd ROM chip is reportedly initially disabled, and can be reportedly enabled by setting [7F48h]=01h (that info doesn't match up with how 7F48h is used by the existing games; unknown if that info is correct/complete).

CX4 CPU Misc

All values are little-endian (opcodes, I/O Ports, cx4rom-ROM-Image, etc).
Call Stack is reportedly 16 levels deep, at least 16bits per level.
Carry Flag is CLEARED on borrow (ie. opposite as on 80x86 CPUs).

CX4 Timings (Unknown)

All opcode & DMA timings are 100% unknown. The CX4 is said to be clocked at 20.000MHz, but this might be internally divided, possibly with different waitstates for different memory regions or different opcodes.
The ROM speed is 2.68Mhz (according to the cartridge header), and 16bit opcodes are passed through 8bit databus (though one may assume that the CX4 contains an opcode cache) (cache might be divided into 200h-byte pages, so, far-jumps to other pages might be slow, maybe/guessed).
The "skip" opcodes are "jumping" to the location after the next opcode (this probably faster than the actual "jmp" opcodes).
After Multiply opcodes one should insert one "nop" (or another instruction that doesn't access the MH or ML result registers).

Reading data bytes from SNES ROM requires some complex timing/handling:

```
612Eh   movb   ext_dta,[ext_ptr]          ;\these 3 opcodes are used to
4000h   inc    ext_ptr                    ; read one byte from [ext_ptr],
1C00h   finish ext_dta                    ;/and to increment ext_ptr by 1
```

The exact meaning of the above opcodes is unknown (which one does what part?).
It is also allowed to use the middle opcode WITHOUT the "prepare/wait" part:

```
4000h   inc    ext_ptr                    ;-increment ext_ptr by 1
```

In that case, "ext_ptr" is incremented, but "ext_dta" should not be used (might be unchanged, or contain garbage, or receive data after some cycles?).

SNES Cart Capcom CX4 - I/O Ports

CX4 I/O Map

```
6000h..6BFFh R/W  CX4RAM (3Kbytes)
6C00h..7F3Fh ?    Unknown/unused
7F40h..7F42h ?/W  DMA source, 24bit SNES LoROM address
7F43h..7F44h ?/W  DMA length, 16bit, in bytes (eg. 0800h = 2Kbytes)
7F45h..7F46h ?/W  DMA destination, 16bit in CX4RAM (6000h = 1st byte)
7F47h        ?/W  DMA start (write 00h to transfer direction SNES-to-CX4)
7F48h        ?/W  Unknown "toggle" (set to 00h/01h, maybe cache load/on/off?)
7F49h..7F4Bh R/W  Program ROM Base, 24bit LoROM addr (028000h in Mega Man)
7F4Ch        ?/W  Unknown (set to 00h or 01h) soft_reset? maybe flush_cache?
7F4Dh..7F4Eh ?/W  Program ROM Instruction Page (PC/200h)
7F4Fh        ?/W  Program ROM Instruction Pointer (PC/2), starts execution
7F50h..7F51h R/W  Unknown, set to 0144h (maybe config flags or waitstates?)
7F52h        R/W  Unknown (set to 00h) hard_reset? maybe force stop?
7F53h..7F5Dh ?    Unknown/unused
7F5Eh        R/?  Status (bit6=busy, set upon [7F47],[7F48],[7F4F] writes)
7F5Fh        ?    Unknown/unused
7F60h..7F69h ?    Unknown/unused (maybe [FFE0..FFE9])
7F6Ah..7F6Bh R/W  SNES NMI Vector       [FFEA..FFEB]
7F6Ch..7F6Dh ?    Unknown/unused (maybe [FFEC..FFED])
7F6Eh..7F6Fh R/W  SNES IRQ Vector       [FFEE..FFEF]
7F70h..7F7Fh ?    Unknown/unused (maybe [FFF0..FFFF])
7F80h..7FAFh R/W  Sixteen 24bit CX4 registers (R0..R15, at 7F80h+N*3)
7FB0h..7FFFh ?    Unknown/unused
8000h..FFFFh R    ROM (32Kbyte LoROM Banks) (disabled when CX4 is busy)
FFExh..FFxxh R/?  Exception Vectors (from above I/O Ports, when CX4 is busy)
```

Exception Vectors
Unknown if these can be manually enabled, or if they are automatically enabled when the CX4 is "busy". In the latter case, they would be REQUIRED to be same as the ROM vectors (else LSB/MSB might be accidently fetched from different locations when busy-flag changes at same time).

SNES Cart Capcom CX4 - Opcodes

CX4 Opcodes (all are 16bit wide)

```
Opcode         Clks NZC Syntax
0000h            ?? ??? nop     ;nop is used as delay after "mul" opcodes
0400h            ?? ??? -
0800h+p0aaaaaaaa ?? ??? jmp   addr/prg_page:addr
0C00h+p0aaaaaaaa ?? ??? jz    addr/prg_page:addr  ;Z=1 (equal)
1000h+p0aaaaaaaa ?? ??? jc    addr/prg_page:addr  ;C=1 (above/equal)
1400h+p0aaaaaaaa ?? ??? js    addr/prg_page:addr  ;N=1 (negative)
1800h            ?? ??? -
1C00h            ?? ??? finish ext_dta
2000h            ?? ??? -
2400h+nn0000000n ?? ??? skip<?/?/nc/c/nz/z/ns/s>  ;skip next opcode
2800h+p0aaaaaaaa ?? ??? call  addr/prg_page:addr
2C00h+p0aaaaaaaa ?? ??? callz addr/prg_page:addr  ;Z=1 (equal)
3000h+p0aaaaaaaa ?? ??? callc addr/prg_page:addr  ;C=1 (above/equal)
3400h+p0aaaaaaaa ?? ??? calls addr/prg_page:addr  ;N=1 (negative)
3800h            ?? ??? -
3C00h            ?? ??? ret
4000h            ?? ??? inc   ext_ptr
4400h            ?? ??? -
4800h+ssoooooooo ?? ??? cmp   <op>,A/A*2/A*100h/A*10000h     ;\
4C00h+ssoooooooo ?? ??? cmp   <imm>,A/A*2/A*100h/A*10000h    ; compare
5000h+ssoooooooo ?? NZC cmp   A/A*2/A*100h/A*10000h,<op>     ;
5400h+ssoooooooo ?? NZC cmp   A/A*2/A*100h/A*10000h,<imm>    ;/
5800h+ss00000000 ?? ??? mov   A,A.?/lsb/lsw/?                ;-sign-expand
5C00h            ?? ??? -
6000h+nnoooooooo ?? ??? mov   A/ext_dta/?/prg_page,<op>
6400h+nnoooooooo ?? ??? mov   A/?/?/prg_page,<imm>
6800h+nnoooooooo ?? ??? movb  ram_dta.lsb/mid/msb/?,cx4ram[<op>]
6C00h+nnoooooooo ?? ??? movb  ram_dta.lsb/mid/msb/?,cx4ram[ram_ptr+<imm>]
7000h+00oooooooo ?? ??? mov   rom_dta,cx4rom[<op>*3]
7400h            ?? ??? -
7800h+0noooooooo ?? ??? mov   prg_page.lsb/msb,<op>
7C00h+0noooooooo ?? ??? mov   prg_page.lsb/msb,<imm>
8000h+ssoooooooo ?? ??C add   A,A/A*2/A*100h/A*10000h,<op>   ;\
8400h+ssoooooooo ?? ?Z? add   A,A/A*2/A*100h/A*10000h,<imm>  ;
8800h+ssoooooooo ?? ??? sub   A,<op>,A/A*2/A*100h/A*10000h   ; add/subtract
8C00h+ssoooooooo ?? ??C sub   A,<imm>,A/A*2/A*100h/A*10000h  ;
9000h+ssoooooooo ?? NZC sub   A,A/A*2/A*100h/A*10000h,<op>   ;
9400h+ssoooooooo ?? NZC sub   A,A/A*2/A*100h/A*10000h,<imm>  ;/
9800h+00oooooooo ?? ??? smul  MH:ML,A,<op>    ;\use NOP or other opcode,
9C00h+00oooooooo ?? ??? smul  MH:ML,A,<imm>   ;/result is signed 48bit
A000h            ?? ??? -
A400h            ?? ??? -
A800h+ssoooooooo ?? ??? xor   A,A/A*2/A*100h/A*10000h,<op>   ;\
AC00h+ssoooooooo ?? ??? xor   A,A/A*2/A*100h/A*10000h,<imm>  ;
B000h+ssoooooooo ?? ?Z? and   A,A/A*2/A*100h/A*10000h,<op>   ; logic
B400h+ssoooooooo ?? ?Z? and   A,A/A*2/A*100h/A*10000h,<imm>  ;
B800h+ssoooooooo ?? ??? or    A,A/A*2/A*100h/A*10000h,<op>   ;
BC00h+ssoooooooo ?? ??? or    A,A/A*2/A*100h/A*10000h,<imm>  ;/
C000h+00oooooooo ?? ??? shr   A,<op>                         ;\
C400h+00oooooooo ?? NZ? shr   A,<imm>                        ;
C800h+00oooooooo ?? ??? sar   A,<op>                         ;
CC00h+00oooooooo ?? N?? sar   A,<imm>                        ; shift/rotate
D000h+00oooooooo ?? ??? ror   A,<op>                         ;
D400h+00oooooooo ?? ??? ror   A,<imm>                        ;
D800h+00oooooooo ?? ??? shl   A,<op>                         ;
DC00h+00oooooooo ?? N?? shl   A,<imm>                        ;/
E000h+00oooooooo ?? ??? mov   <op>,A
E400h            ?? ??? -
E800h+nnoooooooo ?? ??? movb  cx4ram[<op>],ram_dta.lsb/mid/msb/?
EC00h+nnoooooooo ?? ??? movb  cx4ram[ram_ptr+<imm>],ram_dta.lsb/mid/msb/?
F000h+00oooooooo ?? ??? xchg  <op>,A
F400h            ?? ??? -
F800h            ?? ??? -
FC00h            ?? ??? stop          ;stop, and clear Port [FF5E].bit6
```

Opcode "Middle" 2bits (Bit9-8)
Selects different parameters for some opcodes (eg. lsb/mid/msb, as shown in above descriptions).

Opcode Lower 8bits (Bit7-0)
Lower Bits <op>:

```
00h Register A
01h Register MH       ;multiply.result.upper.24bit (MSBs are sign-expanded)
02h Register ML       ;multiply.result.lower.24bit (same for signed/unsigned)
03h Register ext_dta
08h Register rom_dta
0Ch Register ram_dta
13h Register ext_ptr  ;24bit SNES memory address
1Ch Register ram_ptr
2Eh Special  snesrom[ext_ptr] (?)  ;for use by opcode 612Eh only (?)
50h Constant 000000h
51h Constant FFFFFFh
52h Constant 00FF00h
53h Constant FF0000h
54h Constant 00FFFFh
55h Constant FFFF00h
56h Constant 800000h
57h Constant 7FFFFFh
58h Constant 008000h
59h Constant 007FFFh
```

```
  5Ah Constant FF7FFFh
  5Bh Constant FFFF7Fh
  5Ch Constant 010000h
  5Dh Constant FEFFFFh
  5Eh Constant 000100h
  5Fh Constant 00FEFFh
  6xh Register R0..R15, aka Port [7F80h+x*3] ;(x=0h..Fh)
```
Lower Bits <imm>:
```
  nnh Immediate 000000h..0000FFh (unsigned)
```
Lower Bits jump/call<addr>:
```
  nnh Program Counter LSBs (within 256-word page) (absolute, non-relative)
```
Lower Bits skip<cond>:
```
  00h Skip next opcode if selected flag is zero (conditions ?/nc/nz/ns)
  01h Skip next opcode if selected flag is set  (conditions ?/c/z/s)
```
Lower Bits for opcodes that don't use them (uuuuuuuu):
```
  00h Unused, should be zero
```

SNES Cart Capcom CX4 - Functions

CX4 Functions (as contained in Mega Man X2/X3 ROMs)
The CX4 functions are located at SNES address 02:8000-02:9FFF (aka CX4 addresses at PAGE:PC=0000:00..000F:FF with BASE=028000):
```
  PAGE:PC__Function_____________________________
  0000:00  build_oam
  0001:00  scale_tiles  ;<-- (seems to be unused by Mega Man games)
  0002:00  hires_sqrt   ;<-- (seems to be unused by Mega Man games)
  0002:03  sqrt         ;<-- (seems to be unused by Mega Man games)
  0002:05  propulsion
  0002:07  get_sin      ;<-- (seems to be unused by Mega Man games)
  0002:0A  get_cos      ;<-- (seems to be unused by Mega Man games)
  0002:0D  set_vector_length
  0002:10  triangle1
  0002:13  triangle2
  0002:15  pythagorean
  0002:1F  arc_tan
  0002:22  trapeziod
  0002:25  multiply
  0002:2D  transform_coordinates
  0003:00  scale_rotate1
  0005:00  transform_lines
  0007:00  scale_rotate2
  0008:00  draw_wireframe_without_clearing_buffer
  0008:01  draw_wireframe_with_clearing_buffer
  000B:00  disintergrate
  000C:00  wave
  000E:00  test_set_r0_to_00h ;\sixteen 4-word functions,
  ...      ...                ; located at 000E:00+4*(0..15)
  000E:3C  test_set_r0_to_0Fh ;/setting R0 to 00h..0Fh
  000E:40  test_2K_ram_chksum
  000E:54  test_square           ;R1:R2 = R0*R0
  000E:5C  test_immediate_register  ;copy 16 cpu constants to 30h-bytes RAM
  000E:89  test_3K_rom_chksum  ;"immediate_rom"
```
Both Mega Man X2 and X3 are containing 1:1 the same CX4 code (the only two differences are different ROM bank numbers for "Wireframe" vertices):
```
  Mega Man X2:  [0008:3B]="or a,a,28h"  [000A:C4]="mov a,28h"  ;ROM bank 28h
  Mega Man X3:  [0008:3B]="or a,a,08h"  [000A:C4]="mov a,08h"  ;ROM bank 08h
```
That differences apply to the US/Canada versions. There <might> be further differences in Japanese and/or European versions(?)

SNES Cart DSP-n/ST010/ST011 (pre-programmed NEC uPD77C25 CPU) (23 games)

Nintendo DSP-n Chips
The DSP-n chips are 28pin NEC uPD77C25 CPUs with internal ROM/RAM. There are six versions:
```
  DSP-1, DSP-1A, DSP-1B, DSP-2, DSP-3, DSP-4
```
DSP-1 and DSP-1A contain exactly the same Program/Data ROM. DSP-1B contains a bug-fixed DSP1/1A version. DSP2/3/4 contain custom ROMs.

Seta ST010/ST011 Chips
These are 64pin chips, containing a slightly extended NEC uPD77C25 with more ROM and RAM, faster CPU clock.
```
  64pin  SETA ST010 D96050CW-012 (PCB SHVC-1DS0B-01)
  64pin  SETA ST011 D96050CW-013 (PCB same/similar as above?)
```
The onchip RAM is battery-backed and is accessible directly via SNES address bus.

NEC uPD77C25 Specs
SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - Registers & Flags & Overview
SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - ALU and LD Instructions
SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - JP Instructions

Game specific info
SNES Cart DSP-n/ST010/ST011 - List of Games using that chips
SNES Cart DSP-n/ST010/ST011 - BIOS Functions

DSPn/ST010/ST011 Cartridge Header
For DSPn Cartridges:
```
  [FFD6h]=03h..05h   Chipset = DSPn (plus battery present/absent info)
```
For ST010/ST011 Cartridges:
```
  [FFD6h]=F6h   Chipset = Custom (plus battery; for the on-chip RAM)
  [FFD4h]=00h   Last byte of Title=00h (indicate early extended header)
  [FFBFh]=01h   Chipset Sub Type = ST010/ST011
```
Note: The uPD77C25's ROM/RAM aren't counted in the ROM Size, ROM Checksum, SRAM Size (nor Expansion RAM Size) entries. The header (nor extended header) includes no info whether a DSPn game uses a DSP1, DSP2, DSP3, or DSP4, and no info if a ST010/ST011 game uses ST010 or ST011. Ideally, the uPD77C25 ROM-Image should be appended at the end of the SNES ROM-Image. In practice, it's often not there, so there's no way to detect if the game uses this or that uPD77C25 ROM (except for using a list of known Titles or Checksums).

SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - Registers & Flags & Overview

DSP Mapping

```
LoROM Mapping:
DSP       PCB           Mode  ROM RAM Bank       Data (DR)     Status (SR)
DSP1/DSP4 SHVC-1B0N-01  LoROM 1M  -   30h-3Fh    8000h-BFFFh   C000h-FFFFh
DSP2      SHVC-1B5B-01  LoROM 1M  32K 20h-3Fh    8000h-BFFFh   C000h-FFFFh
DSP3      SHVC-1B3B-01  LoROM 1M  8K  20h-3Fh    8000h-BFFFh   C000h-FFFFh
DSP1      SHVC-2B3B-01  LoROM 2M  8K  60h-6Fh    0000h-3FFFh   4000h-7FFFh
ST010     SHVC-1DS0B-01 LoROM 1M  -   60h-6xh    0000h         0001h
ST011     SHVC-1DS0B-?  LoROM 512K-   60h-6xh    0000h         0001h
HiROM Mapping:
DSP   PCB          Mode  ROM RAM Bank            Data (DR)     Status (SR)
DSP1  SHVC-1K0N-01 HiROM  4M   - 00h-1Fh         6000h-6FFFh   7000h-7FFFh
DSP1  SHVC-1K1B-01 HiROM  4M  2K 00h-1Fh         6000h-6FFFh   7000h-7FFFh
DSP1B SHVC-1K1X-01 HiROM  4M  2K 00h-0Fh,20h-2Fh 6000h-6FFFh   7000h-7FFFh
DSP1B SHVC-2K1X-01 HiROM  2M  2K 00h-0Fh,20h-2Fh 6000h-6FFFh   7000h-7FFFh
DSP1B SHVC-2K3X-01 HiROM  2M  8K 00h-0Fh,20h-2Fh 6000h-6FFFh   7000h-7FFFh
SFC-Box:
DSP   PCB          Mode  ROM RAM Bank            Data (DR)     Status (SR)
DSP1? GS 0871-102  <Might have variable LoROM/HiROM mapping supported?>
```

Some of the above PCB names seem to be nonsense (eg. DSP with 2MbyteLoROM hasn't ever been produced, except as prototype board).

SNES I/O Ports

```
Type                 DR               SR               SRAM
DSPn+LoROM (1MB)     30-3F:8000-BFFF  30-3F:C000-FFFF  None
DSPn+LoROM (1MB+RAM) 20-3F:8000-BFFF  20-3F:C000-FFFF  70-7D:0000-7FFF
DSPn+LoROM (2MB+RAM) 60-6F:0000-3FFF  60-6F:4000-7FFF  70-7D:0000-7FFF
DSPn+HiROM           00-1F:6000-6FFF  00-1F:7000-7FFF  20-3F:6000-7FFF ?
DSPn+HiROM (MAD-2)   00-0F:6000-6FFF  00-0F:7000-7FFF  30-3F:6000-7FFF ?
ST010/ST011+LoROM    60-6x:0000       60-6x:0001       68-6F:0000-0FFF
```

All banks in range 00-7F are also mirrored to 80-FF. The "LoROM (2MB)" type wasn't actually produced (but is defined in Nintendo's specs, see book1.pdf page 52).

For ST010/ST011, the RAM is contained in the ST01n chip, and is sized 2Kx16bit, whereas the SNES accesses it as 4Kx8bit (even addresses accessing the LSB, odd ones the MSB of the 16bit words).

Registers

```
DP          8-bit Data RAM Pointer               (ST010/11: 11-bit)
RP          10-bit Data ROM Pointer              (ST010/11: 11-bit)
PC          11-bit Program ROM Counter           (ST010/11: 14-bit)
STACK       11-bit x 4-levels (for call/ret/irq) (ST010/11: 14-bit x 8-level)
K,L         two 16bit registers (multiplier input)
AccA,AccB   two 16bit registers (ALU accumulators) (aka A and B)
FlagA,FlagB two 6bit registers with S1,S0,C,Z,OV1,OV0 flags for AccA/AccB
TR,TRB      two 16bit registers (temporary storage)
SR          16bit status I/O register
DR          parallel I/O data (selectable 8bit/16bit via SR's DRC bit)
SI,SO       serial I/O data (selectable 8bit/16bit via SR's SOC,SIC bits)
```

FlagA/FlagB

```
S0  Sign Flag     (set if result.bit15)
Z   Zero Flag     (set if result=0000h)
C   Carry Flag    (set if carry or borrow)
OV0 Overflow Flag (set if result>+7FFFh or result<-8000h)
S1  Direction of Last Overflow (if OV0 then S1=S0, else S1=unchanged)
OV1 Number of Overflows (0=even, 1=odd) (inverted when OV0 gets set)
```

S0,Z,C,OV0 are "normal" flags as used by various CPUs. S1,OV1 are specials for use with JSA1/JSB1/JNSA1/JNSB1 and JOVA1/JOVB1/JNOVA1/JNOVB1 conditional jump opcodes, or with SGN operand (which equals 8000h-SA1). Examples:

```
or  a,a      ;SA1=A.Bit15 (undocumented)     ;\officially     ;\No Addition
mov l,sgn    ;L=8000h-SA1 (but used by DSP1) ;/SA1=Undefined  ;/
mov a,val1                                                    ;\
add a,val2   ;affect OVA0 (and, if OVA0 set, also SA1)        ; Adding
jnova0 skip0 ;test OVA0                                       ; Two Values
mov a,sgn    ;A=8000h-SA1 (saturate max=+7FFFh, min=-8000h)   ;
skip0:                                                        ;/
;below works with up to three 16bit values,                   ;\
;would also work with hundreds of small 8bit values,          ;
;ie. works if multiple overflows occur in opposite directions ;
;but doesn't work if two overflows occur in same direction)   ; Adding
xor a,a      ;clear OVA1                                      ; More Values
add a,val1   ;no overflow OVA1 yet                            ;
add a,val2   ;this may set OVA1                               ;
add a,val3   ;this may set/reset OVA1                         ;
jnova1 skip1 ;test OVA1 (skip if 0 or 2 overflows occurred)   ;
mov a,sgn    ;A=8000h-SA1 (done if 1 overflow occurred)       ;
skip1:                                                        ;/
```

Note: The JSA1/JSB1/JNSA1/JNSB1 and JOVA1/JOVB1/JNOVA1/JNOVB1 opcodes aren't used by any of the DSP1-DSP4 or ST010-ST011 games. The SGN operand is used only by DSP1/DSP1A/DSP1B (once in conjuntion with JNOVA0, which seems to be ALWAYS skipping the SGN-part, and once in conjunction with an OR opcode, which loads an undocumented value to SA1).

Status Register (SR)

```
15    RQM (R)  Request for Master (0=Busy internally, 1=Request external I/O)
14-13 USF1-0   User's Flags (general purpose)        (0=Low, 1=High)
12    DRS (R)  DR Status (for 16bit DR mode; 2x8bit) (0=Ready, 1=Busy)
11    DMA      Direct Memory Access Mode             (0=Non-DMA, 1=DMA)
10    DRC      DR Control, parallel data length      (0=16bit, 1=8bit)
9     SOC      SO Control, serial data output length (0=16bit, 1=8bit)
8     SIC      SI Control, serial data input length  (0=16bit, 1=8bit)
7     EI       Interrupt Enable                      (0=Disable, 1=Enable)
6-2   N/A (R?) Unused/Reserved  (should be zero) (read=always zero?)
1-0   P1-0     Output to P0,P1 pins (0=Low, 1=High)
```

SR.Bit15-8 are output to D7-0 pins (when /CS=LOW, /RD=LOW, /WR=HIGH, A0=HIGH).
SR.Bit1-0 are always output to P1-0 pins.

RQM

RQM gets set when the uPD77C25 does read/write its DR register, and gets cleared when the remote CPU does complete reading/writing DR (complete: when

DRC=8bit, or when DRC=16bit and DRS=Ready). DRS gets toggled in 16bit mode after each 8bit fragment being read/written by remote CPU (the fragments are transferred LSB first, then MSB).

DMA
The DRQ-pin isn't connected in SNES cartridges (since the DMA protocol isn't compatible with the SNES), so there's no real DMA support. However, the uPD77C25 (or at least the ST011) is fast enough to handle SNES-DMA transfers by software. Software for DSP2 uses the 65C816's block-transfer command (which is a bit slower than DMA, but, like DMA, doesn't use handshaking).

Memory
```
2048 x 24bit Instruction ROM/PROM Opcodes
2048 x 1bit  Instruction PROM Protection Flags (0=Lock, 1=Allow Dumping)
1024 x 16bit Data ROM/PROM
256 x 16bit  Data RAM
```

ROM-Images
DSPn/ST010/ST011 ROM-images consist of the Program ROM followed by the Data ROM. Caution: There are several differently formatted dumps of these ROMs:
```
Oldest Files  10K (DSPn)              --> Big-Endian, 24bit-to-32bit padding
Old Files     8K (DSPn) or 52K (ST01n) --> Big-Endian, raw 24bit opcodes
Newer Files   8K (DSPn) or 52K (ST01n) --> Little-Endian, raw 24bit opcodes
```
Preferred would be the "Newer" format. To detect the endianness: All existing ROMs contain "JRQM $" within first four opcodes (97C00xh, with x=0/4/8/C depending on whether it is 1st/2nd/3rd/4th opcode). Ie. possible cases are:
```
Oldest Files  97h,C0h,0xh,FFh  ;big-endian 24bit, plus FFh-padding byte
Old Files     97h,C0h,0xh      ;big-endian 24bit, without padding
Newer Files   0xh,C0h,97h      ;little-endian 24bit, without padding
```
If the "JRQM $" opcode doesn't exist, best default to "Newer" format (that might happen only with uncommon homebrewn DSP ROMs, not with the original ROMs).
Ideally, the ROM-Image should be attached at the end of the SNES Cartridge ROM-Image (when doing that, best remove any 200h-byte header, since the existing headers don't define if/how to adjust their size-entries for DSPn/ST01n ROMs).

Chips
```
uPD77C25   Mask ROM
uPD77P25   Programmable PROM/UVEPROM
```

uPD77C25 Opocde Encoding
All opcodes are 24bit wide. All opcodes are executed in one clock cycle (at max=8.192MHz clock).

```
 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+--+--+-----+-----------+--+-----+-----------+--+-----------+-----------+
|0 |RT|  P  | ALU opcode|A | DPL |    DPH    |RP|    SRC    |    DST    | ALU
+--+--+-----+-----------+--+-----+-----------+--+-----------+-----+-----+
|1 |0 |    BRCH (jump opcode)    |    NA (11bit Next Address)     |  -  | JP
+--+--+--------------------------+--------------------+-----+-----+-----+
|1 |1 |            ID (16bit Immediate Data)          |  -  |    DST    | LD
+--+--+-----------------------------------------------+-----+-----------+
```

uPD77C20 Opocde Encoding (older pre-77C25 version) (not used in SNES)
All opcodes are 23bit wide. All opcodes are executed in one clock cycle (at max=4.xxxMHz clock).

```
 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+--+--+-----+-----------+--+-----+--------+--+-----------+-----------+
|0 |RT|  P  | ALU opcode|A | DPL |  DPH   |RP|    SRC    |    DST    | ALU
+--+--+-----+-----------+--+-+---+--------+--+-----------+-----+-----+
|1 |0 |  BRCH (jump opcode)  |  NA (9bit Next Address)   |     -     | JP
+--+--+----------------------+--------------------+---+--+-----+-----+
|1 |1 |            ID (16bit Immediate Data)          |- |    DST    | LD
+--+--+-----------------------------------------------+--+-----------+
```
DPH is only 3bit (M0..M7), BRCH is only 8bit (without JDPLN0, JDPLNF opcodes). Data ROM entries are only 13bit wide (sign-expanded to 16bit... or left-shifted to 16bit?). NA is only 9bit. Internal clock is only 4MHz. TRB register is not supported.

SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - ALU and LD Instructions

ALU Instructions (Artithmetic/Logical Unit)
```
23    0   Must be 0 for ALU opcodes
22    RT  Return after ALU (0=No/Normal, 1=Yes/Return from Call/Interrupt)
21-20 P   ALU input P (0=RAM[DP], 1=IDB(SRC), 2=K*L*2/10000h, 3=K*L*2)
19-16 ALU ALU opcode
15    A   ALU input/output Q (0=AccA, 1=AccB)
14-13 DPL Data RAM Pointer DP.3-0 adjust (0=DPNOP, 1=DPINC, 2=DPDEC, 3=DPCLR)
12-9  DPH Data RAM Pointer DP.7-4 adjust (0..0Fh=M0..MF) (XOR by that value)
8     RP  Data ROM Pointer RP.9-0 adjust (0=RPNOP, 1=RPDEC)
7-4   SRC Source (copied to DST, and, for ALU, to "IDB" internal data bus)
3-0   DST Destination (copied from SRC)
```
Allows to combine an ALU operation with memory load & store, DP/RP pointer adjustment, optional RET from CALL, along with the K*L*2 multiplication.

LD Instructions (Load)
```
23    1   Must be 1 for LD opcodes
22    1   Must be 1 for LD opcodes
21-6  ID  16bit Immediate
5-4   -   Reserved (should be zero)
3-0   DST Destination (copied from ID)
```
Load is mainly for initialization purposes & other special cases (normally it's faster load immediates from Data ROM, via SRC=RO in ALU opcodes).

ALU Opcode (Bit19-16)
```
Hex Name      ;Expl.                    S1  S0  Cy  Zf OV1 OV0
00h NOP       ;No operation             -   -   -   -  -   -
01h OR        ;Acc = Acc OR P           sf  sf  0   zf 0   0
02h AND       ;Acc = Acc AND P          sf  sf  0   zf 0   0
03h XOR       ;Acc = Acc XOR P          sf  sf  0   zf 0   0
04h SUB       ;Acc = Acc - P            *   sf  cy  zf *   ov
05h ADD       ;Acc = Acc + P            *   sf  cy  zf *   ov
06h SBB       ;Acc = Acc - P - OtherCy  *   sf  cy  zf *   ov
```

```
07h ADC       ;Acc = Acc + P + OtherCy    *   sf  cy  zf *   ov
08h DEC       ;Acc = Acc - 1              *   sf  cy  zf *   ov
09h INC       ;Acc = Acc + 1              *   sf  cy  zf *   ov
0Ah NOT       ;Acc = Acc XOR FFFFh        sf  sf  0   zf 0   0
0Bh SAR1      ;Acc = Acc/2    ;signed     sf  sf  cy  zf 0   0
0Ch RCL1      ;Acc = Acc*2 + OtherCy      sf  sf  cy  zf 0   0
0Dh SLL2      ;Acc = Acc*4 + 3            sf  sf  0   zf 0   0
0Eh SLL4      ;Acc = Acc*16 + 15          sf  sf  0   zf 0   0
0Fh XCHG      ;Acc = Acc ROL 8            sf  sf  0   zf 0   0
```

ADD/ADC/SUB/SBB/INC/DEC set "S1=sf" and "OV1=OV1 XOR 1" upon overflow (and leave S1 and OV1 both unchanged if no overflow).
OtherCy is the incoming carry flag from other accumulator (ie. Cy from FlagA when using AccB).
Note: "NOT" is called "CMP"=Complement in official syntax; it isn't Compare. "SAR/RCL/SLL" are called "SHR/SHL/SHL" in official syntax; though they aren't "normal" logical shifts. OR/AND/XOR/NOT/SAR/RCL/SLL/XCHG do offially set S1=Undefined (but actually it seems to be S1=sf; required for loopings in "Super Air Diver 2"; which uses OR opcode followed by SGN operand).

Multiplier
After each instruction (namely after any ALU and LD instructions that have changed K or L registers), the hardware computes K*L*2 (signed 16bit*16bit->32bit). Result on overflows (-8000h*-8000h*2) is unknown.

SRC Field (Bit7-4) and DST Field (Bit3-0)

```
Hex   SRC (Source, Bit7-4)           DST (Destination, Bit3-0)
00h   TRB  (Temporary B)             @NON (none)
01h   A    (AccA)                    @A   (AccA)
02h   B    (AccB)                    @B   (AccB)
03h   TR   (Temporary A)             @TR  (Temporary A)
04h   DP   (Data RAM Pointer)        @DP  (Data RAM Pointer)
05h   RP   (Data ROM Pointer)        @RP  (Data ROM Pointer)
06h   RO   (ROM[RP])                 @DR  (parallel I/O port)
07h   SGN  (saturation = 8000h-SA1)  @SR  (status register)
08h   DR   (parallel I/O port)       @SOL (SO serial LSB first)
09h   DRNF (DR without RQM/DRQ)      @SOM (SO serial MSB first)
0Ah   SR   (status register)         @K   (Multiply Factor A)
0Bh   SIM  (SI serial MSB first)     @KLR (K=SRC and L=ROM[RP])
0Ch   SIL  (SI serial LSB first)     @KLM (L=SRC and K=RAM[DP OR 40h])
0Dh   K    (Multiply Factor A)       @L   (Multiply Factor B)
0Eh   L    (Multiply Factor B)       @TRB (Temporary B)
0Fh   MEM  (RAM[DP])                 @MEM (RAM[DP])
```

When not using SRC: specify "NON" in source code, and 00h (a dummy TRB fetch) in binary code.
Following combinations are prohibited in ALU instructions:

```
DST field = @KLR or @KLM combined with SRC field = K or L register
DST field and SRC field specify the same register
P-SELECT field = RAM, DST field = @MEM (for ALU operation)
```

Everything else should be allowed (included ALU with SRC=Acc, eg ADD AccA,AccA)

Variants
The older uPD77C20 doesn't support TRB: SRC=00h is NON (=zero/undefined?), DST=0Eh (and also DST=00h) is @NON. Opcodes are only 23bit wide (strip ALU.Bit11/MSB of DPH, and LD.Bit5/Reserved).

SNES Cart DSP-n/ST010/ST011 - NEC uPD77C25 - JP Instructions

JP Instructions (Jump/Call)

```
23    1   Must be 1 for JP opcodes
22    0   Must be 0 for JP opcodes
21-13 BRC Jump/Call opcode
12-2  NA  11bit Next Address Bit0-10 (000h..7FFh, in 24-bit word steps)
1-0   -   Reserved (should be zero) (ST010/ST011: Bit12-11 of NA)
```

BRCH Opcode (Bit21-13)

```
Binary    Hex  Op      Expl.
000000000 000h JMPSO * Unconditional jump to SO register
100000000 100h JMP     Unconditional jump 0000h..1FFFh
100000001 101h JMP   * Unconditional jump 2000h..3FFFh
101000000 140h CALL    Unconditional call 0000h..1FFFh  ;\return via RT-bit
101000001 141h CALL  * Unconditional call 2000h..3FFFh  ;/in ALU opcodes
010000000 080h JNCA    CA = 0         ;\
010000010 082h JCA     CA = 1         ; carry flag of AccA/AccB
010000100 084h JNCB    CB = 0         ;
010000110 086h JCB     CB = 1         ;/
010001000 088h JNZA    ZA = 0         ;\
010001010 08Ah JZA     ZA = 1         ; zero flag of AccA/AccB
010001100 08Ch JNZB    ZB = 0         ;
010001110 08Eh JZB     ZB = 1         ;/
010010000 090h JNOVA0  OVA0 = 0       ;\
010010010 092h JOVA0   OVA0 = 1       ; overflow flag for last operation
010010100 094h JNOVB0  OVB0 = 0       ;
010010110 096h JOVB0   OVB0 = 1       ;/
010011000 098h JNOVA1  OVA1 = 0       ;\
010011010 09Ah JOVA1   OVA1 = 1       ; overflow flag for last 3 operations
010011100 09Ch JNOVB1  OVB1 = 0       ; (set if 1 or 3 overflows occurred)
010011110 09Eh JOVB1   OVB1 = 1       ;/
010100000 0A0h JNSA0   SA0 = 0        ;\
010100010 0A2h JSA0    SA0 = 1        ; sign bit (ie. Bit15) of AccA/AccB
010100100 0A4h JNSB0   SB0 = 0        ;
010100110 0A6h JSB0    SB0 = 1        ;/
010101000 0A8h JNSA1   SA1 = 0        ;\
010101010 0AAh JSA1    SA1 = 1        ; extra sign bit ("Bit16")
010101100 0ACh JNSB1   SB1 = 0        ; indicating direction of overflows
010101110 0AEh JSB1    SB1 = 1        ;/
010110000 0B0h JDPL0   DPL = 00h      ;\
010110001 0B1h JDPLN0  DPL <> 00h     ; lower 4bit of DP (Data RAM Pointer)
010110010 0B2h JDPLF   DPL = 0Fh      ;
010110011 0B3h JDPLNF  DPL <> 0Fh     ;/
010110100 0B4h JNSIAK  SI ACK = 0     ;\
010110110 0B6h JSIAK   SI ACK = 1     ; serial I/O port (SI/SO serial in/out)
010111000 0B8h JNSOAK  SO ACK = 0     ;
010111010 0BAh JSOAK   SO ACK = 1     ;/
```

```
010111100 0BCh JNRQM   RQM = 0        ;\parallel I/O port (DR data register)
010111110 0BEh JRQM    RQM = 1        ;/
```

Jump addresses should be specified 24bit word units (not in byte units).
(*) Opcodes 000h,101h,141h supported on ST010/ST011 only. On that CPU, PC.bit13 can be manipulated by unconditional jump/call/ret (whilst conditional jumps affect only PC.bit12-0).

Reset (Vector 000h)
Reset is triggered when RST pin is high, and does set PC=000h, FlagA=00h, FlagB=00h, SR=0000h, DRQ=0, SORQ=0, SI.ACK=0, SO.ACK=0, and RP=3FFh. Other registers and Data RAM are left unchanged.

Interrupts (Vector 100h)
Interrupts are triggered on raising edge of INT pin. If interrupts are enabled (in SR register), the CPU jumps to address 100h, and pushes PC on stack, the interrupts are NOT automatically disabled.

Variants
The older uPD77C20 doesn't support JDPLN0 and JDPLNF. Opcodes are only 23bit wide (strip JP.Bit13/LSB of BRCH). And PC/NA is only 9bit wide (replace JP.Bit3-2 by Reserved).

SNES Cart DSP-n/ST010/ST011 - List of Games using that chips

Games using DSPn/ST01n chips
The DSP-1/1A/1B is used by around 16..19 games:

```
Ace Wo Nerae! 3D Tennis (DSP-1A) (1993) Telenet Japan (JP)
Armored Trooper Votoms: The Battling Road (1993) Takara (JP)
Ballz 3D, and 3 Jigen Kakutou Ballz (DSP-1B) (1994) PF Magic/Accolade (NA)
Battle Racers (1995) Banpresto (JP)
Bike Daisuki! Hashiriya Kon - Rider's Spirits (1994) Genki/NCS (JP)
Final Stretch (1993) Genki/LOZC (JP)
Korean League (aka Hanguk Pro Yagu) (1993) Jaleco (KO)
Lock-On / Super Air Diver (1993) Vic Tokai
Michael Andretti's Indy Car Challenge (1994) Genki/Bullet-Proof (NA) (JP)
Pilotwings (1991) Nintendo EAD (NA) (JP) (EU) (DSP-1) (visible DSP1 glitch)
Shutokou Battle'94: K.T. Drift King (1994) Genki/Bullet-Proof (JP)
Shutokou Battle 2: Drift King K.T. & M.B. (1995) Genki/Bullet-Proof (JP)
Super 3D Baseball (?) (is that same as Super Bases Loaded 2 ?)
Super Air Diver 2 (1995) Asmik (JP)
Super Bases Loaded 2 (1994) Jaleco (NA) (JP)
Super F1 Circus Gaiden (1995) Nichibutsu (JP)
Super Mario Kart (DSP-1/DSP-1B) (1992) Nintendo EAD (NA) (JP) (EU)
Suzuka 8 Hours (1993) Namco (NA) (JP)
Touge Densetsu: Saisoku Battle (1996) Genki/Bullet-Proof Software (JP) ?
```

The other five versions are used by only one game each:

```
DSP-2: Dungeon Master (DSP-2) (1992) FTL Games/JVC Victor (JP)
DSP-3: SD Gundam GX (DSP-3) (1994) BEC/Bandai (JP)
DSP-4: Top Gear 3000 (DSP-4) (1995) Gremlin Interactive/Kemco (NA) (JP) (EU)
ST010: F1 Race of Champions / Exhaust Heat II (1993) SETA Corp. (NA) (JP)
ST011: Hayazashi Nidan Morita Shogi (1993) Random House/SETA Corp. (JP)
```

SNES Cart DSP-n/ST010/ST011 - BIOS Functions

DSP1 Commands
When requesting data from an external device the DSP is oblivious to the type of operation that occurs to the Data Register. Writing to the Data register will update the contents of the register and allow the DSP to continue execution. Reading from the Data Register will also allow the DSP to continue execution. On completion of a valid command the Data Register should contain the value 0x80. This is to prevent a valid command from executing should a device read past the end of output.

```
00h  16-bit Multiplication
10h  Inverse Calculation
20h  16-bit Multiplication
01h  Set Attitude A
11h  Set Attitude B
21h  Set Attitude C
02h  Projection Parameter Setting
03h  Convert from Object to Global Coordinate A
13h  Convert from Object to Global Coordinate B
23h  Convert from Object to Global Coordinate C
04h  Trigonometric Calculation
14h  3D Angle Rotation
06h  Object Projection Calculation
08h  Vector Size Calculation
18h  Vector Size Comparison
28h  Vector Absolute Value Calculation (bugged) (fixed in DSP1B)
38h  Vector Size Comparison
0Ah  Raster Data Calculation
0Bh  Calculation of Inner Product with the Forward Attitude A and a Vector
1Bh  Calculation of Inner Product with the Forward Attitude B and a Vector
2Bh  Calculation of Inner Product with the Forward Attitude C and a Vector
0Ch  2D Coordinate Rotation
1Ch  3D Coordinate Rotation
0Dh  Convert from Global to Object Coordinate A
1Dh  Convert from Global to Object Coordinate B
2Dh  Convert from Global to Object Coordinate C
0Eh  Coordinate Calculation of a selected point on the Screen
0Fh  Test Memory Test
1Fh  Test Transfer DATA ROM
2Fh  Test ROM Version (0100h=DSP1/DSP1A, 0101h=DSP1B)
```

Command 28h is bugged in DSP1/DSP1A (fixed in DSP1B) bug is evident in Pilotwings (Plane Demo).

DSP2 Commands (Dungeon Master)
This chip does - amazingly - assist 3D labyrinth drawing operations that are normally implemented on ZX81 computers.

```
01h  Convert Bitmap to Bitplane Tile
03h  Set Transparent Color
05h  Replace Bitmap using Transparent Color
```

```
06h  Reverse Bitmap
07h  Add
08h  Subtract
09h  Multiply (bugged) (used in Dungeon Master japanese/v1.0)
0Dh  Scale Bitmap
0Fh  Process Command (dummy NOP command for re-synchronisation)
```

DSP3 Commands (SD Gundam GX)
The DSP functions inherently similiar to the DSP1 with respect to command parsing and execution. On completion of a valid command the Data Register should contain the value 0x80.

```
02h  Unknown
03h  Calculate Cell Offset
06h  Set Board Dimensions
07h  Calculate Adjacent Cell
18h  Convert Bitmap to Bitplane
38h  Decode Shannon-Fano Bitstream (USF1 bit in SR register = direction)
1Eh  Calculate Path of Least Travel
3Eh  Set Start Cell
0Fh  Test Memory Test
1Fh  Test Transfer DATA ROM
2Fh  Test ROM Version (0300h)
```

DSP4 Commands (Top Gear 3000)
On completion of a valid command the Data Register should contain the value 0xffff. This is to prevent a valid command from executing should an external device read past the end of output. Unlike previous DSP programs, all data transfers are 16-bit.

```
xxh  Unknown
```

ST010 Commands
Commands are executed on the ST-0010 by writing the command to 0x0020 and setting bit 7 of 0x0021. Bit 7 of 0x0021 will stay set until the Command has completed, at which time output data will be available. See individual commands for input and output parameter addresses.

```
01h  Unknown Command
02h  Sort Driver Placements
03h  2D Coordinate Scale
04h  Unknown Command
05h  Simulated Driver Coordinate Calculation
06h  Multiply
07h  Raster Data Calculation
08h  2D Coordinate Rotation
```

The ST010 BIOS functions are more or less useless and don't increase the performance or quality of the game (the only feature that is <really> used is the battery-backed on-chip RAM, aside from that, the powerful chip is a waste of resources). Note: The ST010 is also used in "Twin Eagle II" (arcade game, not a SNES game).

ST011 Commands

```
xxh  Unknown (japanese chess engine)
```

SNES Cart Seta ST018 (pre-programmed ARM CPU) (1 game)

Seta ST018 - 160pin SETA D6984 ST018 chip (PCB SHVC-1DE3B-01)
The chip is used by a single game only:

```
Hayazashi Nidan Morita Shogi 2 (ST018) (1995) Random House/SETA Corp. (JP)
```

Note: The ST018 is occassionally referred to as ST0018 (the correct <chip> name is ST018, the double-zero variant is found on the PCB's text layer).
I/O Ports are at 3800h..3804h and (maybe) also at FF40h..FF63h (the latter ones might be constants in ROM, or read-only I/O ports; if they are I/O ports: unknown how that'd affect the ROM-checksum).

```
3800h.R  Data (from ST018 to SNES) (when STAT.4=0 and STAT.0=1)
3802h.W  Data (from SNES to ST018) (when STAT.4=0)
3802h.R  Ack (dummy read used to acknowledge something)
3804h.W  Control (00h=Normal, 01h=HardReset, FFh=SoftReset?)
3804h.R  Status
```

ST018 Status Bits

```
0 Ready (0=Busy, 1=Ready) (prior to DATA read from [3800])
1 Unknown/Unused
2 Ready (0=Busy, 1=Ready) (checked before/after CMD AB) (and after CMD AE)
3 Unknown/Unused
4 Transfer Request (0=Okay/Ready/RequestDataIn/Out, 1=Fail/Busy/NoRequest)
5 Unknown/Unused
6 Ready (0=Busy, 1=Ready) (used only when [FF41]<>00h)
7 Ready (0=Busy, 1=Ready) (used alongside softreset and hardreset)
```

ST018 commands

```
A3  Debug (send 4 byte (32bit) address, then receive 128 bytes)
AA  UploadBoardAndSomethingElse (send 9x9 plus 16 bytes)
AB  unknown (send 1 byte) (02h) (maybe len for command AD)
AD  unknown (receive 2 bytes)
AE  unknown (no parameters)
AF  unknown (receive 1 byte) (len for command B0)
B0  unknown (receive LEN bytes) (LEN from command AF)
B1  unknown (send 2 bytes)
B2  unknown (send 2 bytes)
B3  unknown (receive 1 byte)
B4  unknown (receive 1 byte)
B5  unknown (receive 1 byte)
B6  unknown (receive 1 byte)
F1  Status/Test (if response.bit2=1, receive 2 error bytes)
F2  Status/Test (if response<>00h, receive 2 error bytes)
F3  Debug (receive 128Kbytes from 0000:0000..0001:FFFF, for HEX-DUMP display)
F4  Debug (receive 32Kbytes from A000:0000..A000:7FFF, for HEX-DUMP display)
```

Command F3 allows to dump the ST018 BIOS, Command F4 probably dumps its RAM.

SNES Cart OBC1 (OBJ Controller) (1 game)

The OBC1 is a 80pin OBJ Controller chip from Nintendo, used by only one game:

```
Metal Combat: Falcon's Revenge (1993) Intelligent Systems/Nintendo
(Note: the game also requires a Super Scope lightgun)
```

OBC1 I/O Ports
```
7FF0h OAM Xloc = [Base+Index*4+0]  (R/W)
7FF1h OAM Yloc = [Base+Index*4+1]  (R/W)
7FF2h OAM Tile = [Base+Index*4+2]  (R/W)
7FF3h OAM Attr = [Base+Index*4+3]  (R/W)
7FF4h OAM Bits = [Base+Index/4+200h].Bit((Index AND 3)*2+0..1) (R?/W)
7FF5h Base for 220h-byte region (bit0: 0=7C00h, 1=7800h)
7FF6h Index (OBJ Number) (0..127)
7FF7h Unknown (set to 00h or 0Ah) (maybe SRAM vs I/O mode select)
```
Other bytes at 6000h..7FFFh contain 8Kbyte battery-backed SRAM (of which, 7800h..7A1Fh and 7C00h..7E1Fh can be used as OBJ workspace).

Notes
Port 7FF0h-7FF3h/7FF5h are totally useless. Port 7FF4h/7FF6h are eventually making it slightly easier to combine the 2bit OAM fragments, though putting a huge 80pin chip into the cartridge for merging 2bit fragments is definetly overcomplicated.
As far as known, the Index isn't automatically incremented. Port 7FF4h does read-modify-write operations which may involve timing restrictions (?), or, modify-write (when prefetching data on 7FF6h writes) which may come up with out-dated-prefetch effects.
Reading from 7FF4h does reportedly return the desired BYTE, but WITHOUT isolating & shifting the desired BITS into place?
Setting Index bits7+5 does reportedly enable SRAM mapping at 6000h..77FFh?
ROM is reportedly mapped to bank 00h..3Fh, and also to bank 70h..71h? Maybe that info just refers to SRAM not being mapped to that region (as it'd be in some other LoROM cartridges).

PCB "SHVC-2E3M-01"
Contains six chips and a battery. The chips are: Two 1MB ROMs, MAD-1, OBC1, CIC, 8K SRAM. All chips (except MAD-1) are SMD chips.

SNES Cart S-DD1 (Data Decompressor) (2 games)

The S-DD1 is a 100pin Data Decompression chip, used by only two games:
```
Star Ocean (6MB ROM, 8KB RAM) (1996) tri-Ace/Enix (JP)
Street Fighter Alpha 2 (4MB ROM, no RAM) (1996) Capcom (NA) (JP) (EU)
```

S-DD1 Decompression Algorithm
SNES Cart S-DD1 Decompression Algorithm

S-DD1 I/O Ports
```
4800h  DMA Enable 1 (bit0..7 = DMA 0..7) (unchanged after DMA)
4801h  DMA Enable 2 (bit0..7 = DMA 0..7) (automatically cleared after DMA)
4802h  Unknown   ;\set to 0000h by Star Ocean (maybe SRAM related)
4803h  Unknown   ;/unused by Street Fighter Alpha 2
4804h  ROM Bank for C00000h-CFFFFFh (in 1MByte units)
4805h  ROM Bank for D00000h-DFFFFFh (in 1MByte units)
4806h  ROM Bank for E00000h-EFFFFFh (in 1MByte units)
4807h  ROM Bank for F00000h-FFFFFFh (in 1MByte units)
<DMA>  DMA from ROM returns Decompressed Data (originated at DMA start addr)
```

S-DD1 Memory Map
```
???-???          SRAM (if any)
008000h-00FFFFh  Exception Handlers, mapped in LoROM-fashion (ROM 0..7FFFh)
C00000h-CFFFFFh  ROM (mapped via Port 4804h) (in HiROM fashion)
D00000h-DFFFFFh  ROM (mapped via Port 4805h) (in HiROM fashion)
E00000h-EFFFFFh  ROM (mapped via Port 4806h) (in HiROM fashion)
F00000h-FFFFFFh  ROM (mapped via Port 4807h) (in HiROM fashion)
```

S-DD1 PCBs
```
SHVC-1NON-01  CartSlotPin59 not connected (no C12 capacitor on PA1 pin)
SHVC-1NON-10  Strange revision (capacitor C12 between PA1 and GND)
SNSP-1NON-10  PAL version (S-DD1.Pin82 wired to ... VCC?) (also with C12)
SHVC-LN3B-01  Version with additional SRAM for Star Ocean
```
The 1NON board contains only two chips (100pin D-DD1 and 44pin ROM), the CIC function is included in the S-DD1, whereas Pin82 does probably select "PAL/NTSC" CIC mode.
The LN3B-board contains five chips (two 44pin ROMs, S-DD1, 8Kx8bit SRAM, and a MM1026AF battery controller).

S-DD1 Pinouts
```
1-81   Unknown
82     PAL/NTSC (for CIC mode)
83-100 Unknown
```

SNES Cart S-DD1 Decompression Algorithm

decompress_init(src)
```
input=[src], src=src+1
if (input AND C0h)=00h then num_planes = 2
if (input AND C0h)=40h then num_planes = 8
if (input AND C0h)=80h then num_planes = 4
if (input AND C0h)=C0h then num_planes = 0
if (input AND 30h)=00h then high_context_bits=01c0h, low_context_bits=0001h
if (input AND 30h)=10h then high_context_bits=0180h, low_context_bits=0001h
if (input AND 30h)=20h then high_context_bits=00c0h, low_context_bits=0001h
if (input AND 30h)=30h then high_context_bits=0180h, low_context_bits=0003h
input=(input SHL 11) OR ([src+1] SHL 3), src=src+1, valid_bits=5
for i=0 to 7 do bit_ctr[i]=00h, prev_bits[i]=0000h
for i=0 to 31 do context_states[i]=00h, context_MPS[i]=00h
plane=0, yloc=0, raw=0
```

decompress_byte(src,dst)
```
if num_planes=0
  for plane=0 to 7 do GetBit(plane)
  [dst]=raw, dst=dst+1
else if (plane AND 1)=0
  for i=0 to 7 do GetBit(plane+0), GetBit(plane+1)
  [dst]=prev_bits[plane] AND FFh, dst=dst+1, plane=plane+1
```

```
  else
    [dst]=prev_bits[plane] AND FFh, dst=dst+1, plane=plane-1
    yloc=yloc+1, if yloc=8 then yloc=0, plane = (plane+2) AND (num_planes-1)
```

GetBit(plane)

```
  context = (plane AND 1) SHL 4
  context = context OR ((prev_bits[plane] AND high_context_bits) SHR 5)
  context = context OR (prev_bits[plane] AND low_context_bits)
  pbit=ProbGetBit(context)
  prev_bits[plane] = (prev_bits[plane] SHL 1) + pbit
  if num_planes=0 then raw = (raw SHR 1)+(pbit SHL 7)
```

ProbGetBit(context)

```
  state=context_states[context]
  code_size=EvolutionCodeSize[state]
  if (bit_ctr[code_size] AND 7Fh)=0 then
    bit_ctr[code_size]=GetCodeword(code_size)
  pbit=context_MPS[context]
  bit_ctr[code_size] = bit_ctr[code_size]-1
  if bit_ctr[code_size]=00h    ;"GolombGetBit"
    context_states[context]=EvolutionLpsNext[state]
    pbit=pbit XOR 1
    if state<2 then context_MPS[context]=pbit
  else if bit_ctr[code_size]=80h
    context_states[context]=EvolutionMpsNext[state]
  return pbit
```

GetCodeword(code_size)

```
  if valid_bits=0 then input=input OR [src], src=src+1, valid_bits=8
  input=input SHL 1, valid_bits=valid_bits-1
  if (input AND 8000h)=0 return 80h+(1 SHL code_size)
  tmp=((input SHR 8) AND 7Fh) OR (7Fh SHR code_size)
  input=input SHL code_size, valid_bits=valid_bits-code_size
  if valid_bits<0 then
    input=input OR (([src] SHL (-valid_bits))
    src=src+1, valid_bits=valid_bits+8
  return RunTable[tmp]
```

EvolutionCodeSize[0..32]

```
  0 , 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3
  4 , 4, 5, 5, 6, 6, 7, 7, 0, 1, 2, 3, 4, 5, 6, 7
```

EvolutionMpsNext[0..32]

```
  25, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17
  18,19,20,21,22,23,24,24,26,27,28,29,30,31,32,24
```

EvolutionLpsNext[0..32]

```
  25, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15
  16,17,18,19,20,21,22,23, 1, 2, 4, 8,12,16,18,22
```

RunTable[0..127]

```
  128, 64, 96, 32, 112, 48, 80, 16, 120, 56, 88, 24, 104, 40, 72, 8
  124, 60, 92, 28, 108, 44, 76, 12, 116, 52, 84, 20, 100, 36, 68, 4
  126, 62, 94, 30, 110, 46, 78, 14, 118, 54, 86, 22, 102, 38, 70, 6
  122, 58, 90, 26, 106, 42, 74, 10, 114, 50, 82, 18,  98, 34, 66, 2
  127, 63, 95, 31, 111, 47, 79, 15, 119, 55, 87, 23, 103, 39, 71, 7
  123, 59, 91, 27, 107, 43, 75, 11, 115, 51, 83, 19,  99, 35, 67, 3
  125, 61, 93, 29, 109, 45, 77, 13, 117, 53, 85, 21, 101, 37, 69, 5
  121, 57, 89, 25, 105, 41, 73,  9, 113, 49, 81, 17,  97, 33, 65, 1
```

SNES Cart SPC7110 (Data Decompressor) (3 games)

The SPC7110 (full name "SPC7110F0A" or "SPC7110Foa") is a 100pin Data Decompression chip from Epson/Seiko, used only by three games from Hudson soft:

```
  Far East of Eden Zero (with RTC-4513) (1995) Red Company/Hudson Soft (JP)
  Momotaro Dentetsu Happy (1996) Hudson Soft (JP)
  Super Power League 4 (1996) Hudson Soft (JP)
```

XXX add info from byuu's "spc7110-mcu.txt" file.

Pinouts

SNES Cart SPC7110 Memory and I/O Map

Memory Map

```
  4800h..4842h      SPC7110 I/O Ports
  6000h..7FFFh      Battery-backed SRAM (8K bytes, in all 3 games)
  8000h..FFFFh      Exception Handlers (Program ROM offset 8000h..FFFFh)
  C00000h..CFFFFFh  Program ROM (1MByte) (HiROM)
  D00000h..DFFFFFh  Data ROM (1MByte-fragment mapped via Port 4831h)
  E00000h..EFFFFFh  Data ROM (1MByte-fragment mapped via Port 4832h)
  F00000h..FFFFFFh  Data ROM (1MByte-fragment mapped via Port 4833h)
```

I/O Ports and SRAM are probably mirrored to banks 00h-3Fh and 80h-BFh.
Program/Data ROM is probably mirrored to 400000h-7FFFFFh, the upper 32K fragments of each 64K bank probably also to banks 00h-3Fh and 80h-BFh.

Reportedly (probably nonsense?)
"data decompressed from data rom by spc7110 mapped to $50:0000-$50:FFFF".
That info would imply that "decompressed data" Port 4800h is mirrored to 500000h-5FFFFFh (though more likely, the "un-decompressed data" is mirrored from D00000h-DFFFFFh).

ROM-Image Format
The existing SPC7110 games are 2MB, 3MB, 5MB in size. Stored like so:
```
000000h..0FFFFFh  Program ROM (1MByte) (HiROM)
100000h..xFFFFFh  Data ROM (1MByte, 2MByte, or 4MByte max)
```
Observe that the SPC7110 ROM checksums at [FFDCh..FFDFh] are calculated unconventionally: 3MB/5MB aren't "rounded-up" to 4MB/8MB. Instead, 3MB is checksummed twice (rounded to 6MB). 2MB/5MB are checksummed as 2MB/5MB (without rounding).

Data ROM Decompression Ports
```
4800h --  Decompressed Data Read
4801h 00  Compressed Data ROM Directory Base, bit0-7
4802h 00  Compressed Data ROM Directory Base, bit8-15
4803h 00  Compressed Data ROM Directory Base, bit16-23
4804h 00  Compressed Data ROM Directory Index
4805h 00  Decompressed Data RAM Target Offset, bit0-7    OFFSET IN BANK $50
4806h 00  Decompressed Data RAM Target Offset, bit8-15   OFFSET IN BANK $50
4807h 00  Unknown ("DMA Channel for Decompression")
4808h 00  Unknown ("C r/w option, unknown")
4809h 00  Decompressed Data Length Counter, bit0-7
480Ah 00  Decompressed Data Length Counter, bit8-15
480Bh 00  Unknown ("Decompression Mode")
480Ch 00  Decompression Status (bit7: 0=Busy/Inactive, 1=Ready/DataAvailable)
```
Direct Data ROM Access
```
4810h 00  Data ROM Read from [Base] or [Base+Offs], and increase Base or Offs
4811h 00  Data ROM Base, bit0-7   (R/W)
4812h 00  Data ROM Base, bit8-15  (R/W)
4813h 00  Data ROM Base, bit16-23 (R/W)
4814h 00  Data ROM Offset, bit0-7   ;\optionally Base=Base+Offs
4815h 00  Data ROM Offset, bit8-15  ;/on writes to both of these registers
4816h 00  Data ROM Step, bit0-7
4817h 00  Data ROM Step, bit8-15
4818h 00  Data ROM Mode
481Ah 00  Data ROM Read from [Base+Offset], and optionally set Base=Base+Offs
```
Unsigned Multiply/Divide Unit
```
4820h 00  Dividend, Bit0-7 / Multiplicand, Bit0-7
4821h 00  Dividend, Bit8-15 / Multiplicand, Bit8-15
4822h 00  Dividend, Bit16-23
4823h 00  Dividend, Bit24-31
4824h 00  Multiplier, Bit0-7
4825h 00  Multiplier, Bit8-15, Start Multiply on write to this register
4826h 00  Divisor, Bit0-7
4827h 00  Divisor, Bit8-15, Start Division on write to this register
4828h 00  Multiply/Divide Result, Bit0-7
4829h 00  Multiply/Divide Result, Bit8-15
482Ah 00  Multiply/Divide Result, Bit16-23
482Bh 00  Multiply/Divide Result, Bit24-31
482Ch 00  Divide Remainder, Bit0-7
482Dh 00  Divide Remainder, Bit8-15
482Eh 00  Multiply/Divide Reset  (write = reset 4820h..482Dh) (write 00h)
482Fh 00  Multiply/Divide Status (bit7: 0=Ready, 1=Busy)
```
Memory Mapping
```
4830h 00  SRAM Chip Enable/Disable (bit7: 0=Disable, 1=Enable)
4831h 00  Data ROM Bank for D00000h-DFFFFFh (1MByte, using HiROM mapping)
4832h 01  Data ROM Bank for E00000h-EFFFFFh (1MByte, using HiROM mapping)
4833h 02  Data ROM Bank for F00000h-FFFFFFh (1MByte, using HiROM mapping)
4834h 00  SRAM Bank Mapping?, workings unknown
```
Real-Time Clock Ports (for external RTC-4513)
```
4840h 00  RTC Chip Enable/Disable (bit0: 0=Disable, 1=Enable)
4841h --  RTC Command/Index/Data Port
4842h --  RTC Ready Status
```

SNES Cart SPC7110 Decompression I/O Ports

4800h - Decompressed Data Read
Reading from this register returns one decompressed byte, and does also decrease the 16bit length counter [4809h] by one.

4801h - Compressed Data ROM Directory Base, bit0-7
4802h - Compressed Data ROM Directory Base, bit8-15
4803h - Compressed Data ROM Directory Base, bit16-23
4804h - Compressed Data ROM Directory Index
Selects a directory entry in Data ROM at [Base+Index*4]. Each entry is 4-bytes in size:
```
Byte0  Decompression Mode (00h,01h,02h)
Byte1  Compressed Data ROM Source Pointer, bit16-23  ;\ordered as so
Byte2  Compressed Data ROM Source Pointer, bit8-15   ; (ie. big-endian)
Byte3  Compressed Data ROM Source Pointer, bit0-7    ;/
```

4805h - Decompressed Data RAM Target Offset, bit0-7 OFFSET IN BANK $50
4806h - Decompressed Data RAM Target Offset, bit8-15 OFFSET IN BANK $50
Reportedly: Destination address in bank 50h, this would imply that the SPC7110 chip contains around 64Kbytes on-chip RAM, which is probably utmost nonsense.
Or, reportedly, too: Causes the first "N" decompressed bytes to be skipped, before data shows up at 4800h. That sounds more or less reasonable. If so, unknown if the hardware does decrement the offset value?

4807h - DMA Channel for Decompression
Unknown. Reportedly "DMA CHANNEL FOR DECOMPRESSION, set to match snes dma channel used for compressed data". That info seems to be nonsense; the registers seems to be always set to 00h, no matter if/which DMA channel is used.

4808h - C r/w option, unknown
Unknown. Reportedly "C r/w option, unknown".

4809h - Decompressed Data Length Counter, bit0-7
480Ah - Decompressed Data Length Counter, bit8-15
This counter is decremented on reads from [4800h]. One can initialize the counter before decompression & check its value during decompression. However, this doesn't seem to be required hardware-wise; the decompression seems to be working endless (as long as software reads [4800h]), and doesn't seem to "stop" when the length counter becomes zero.

480Bh - Decompression Mode
Reportedly:
```
00 - manual decompression, $4800 is used to read directly from the data rom

02 - hardware decompression, decompressed data is mapped to $50:0000,
     $4800 can be used to read sequentially from bank $50
```

480Ch - Decompression Status (bit7: 0=Busy/Inactive, 1=Ready/DataAvailable)
Reportedly:
```
DECOMPRESSION FINISHED STATUS:
high bit set = done, high bit clear = processing,
cleared after successful read,
high bit is cleared after writing to $4806,
$4809/A is set to compressed data length
---
decompression mode is activated after writing to $4806
and finishes after reading the high bit of $480C
```

SNES Cart SPC7110 Direct Data ROM Access

4810h Data ROM Read from [Base] or [Base+Offs], and increase Base or Offs
481Ah Data ROM Read from [Base+Offset], and optionally set Base=Base+Offs
Reportedly,
Testing leads to believe that the direct ROM read section starts out as inactive.
One of the ways to activate direct reads is to write a non-zero value to $4813.
No other action need be taken. You can write a non-zero value and immediately
write a zero to it and that's OK. The order of writes to $4811/2/3 don't
seem to matter so long as $4813 has been written to once with a non-zero
value. There may be a way to deactivate the direct reads again (maybe a
decompression cycle?).
There appears to be another way to activate direct reads that is more complex.

4811h Data ROM Base, bit0-7 (R/W)
4812h Data ROM Base, bit8-15 (R/W)
4813h Data ROM Base, bit16-23 (R/W)

4814h Data ROM Offset, bit0-7 ;\optionally Base=Base+Offs
4815h Data ROM Offset, bit8-15 ;/on writes to both of these registers
4816h Data ROM Step, bit0-7
4817h Data ROM Step, bit8-15

4818h Data ROM Mode
```
0   Select Step   (for 4810h) (0=Increase by 1, 1=Increase by "Step" Value)
1   Enable Offset (for 4810h) (0=Disable/Read Ptr, 1=Enable/Read Ptr+Offset)
2   Expand Step from 16bit to 24bit           (0=Zero-expand, 1=Sign-expand)
3   Expand Offset from 8bit?/16bit to 24bit   (0=Zero-expand, 1=Sign-expand)
4   Apply Step (after 4810h read)    (0=On 24bit Pointer, 1=On 16bit Offset)
5-6 Special Actions (see below)
7   Unused (should be zero)
```
Special Actions:
```
0=No special actions
1=After Writing $4814/5 --> 8 bit offset addition using $4814
2=After Writing $4814/5 --> 16 bit offset addition using $4814/5
3=After Reading $481A   --> 16 bit offset addition using $4814/5
```
Reportedly,
```
4818 write: set command mode,
4818 read: performs action instead of returning value, unknown purpose
command mode is loaded to $4818 but only set after writing to both $4814
and $4815 in any order
$4811/2/3 may increment on a $4810 read depending on mode byte)
$4814/$4815 is sometimes incremented on $4810 reads (depending on mode byte)
```
Note: the data rom command mode is activated only after registers $4814 and $4815 have been written to, regardless of the order they were written to

4831h Data ROM Bank for D00000h-DFFFFFh (1MByte, using HiROM mapping)
4832h Data ROM Bank for E00000h-EFFFFFh (1MByte, using HiROM mapping)
4833h Data ROM Bank for F00000h-FFFFFFh (1MByte, using HiROM mapping)

4830h SRAM Chip Enable/Disable (bit7: 0=Disable, 1=Enable)
4834h SRAM Bank Mapping?, workings unknown

SNES Cart SPC7110 Multiply/Divide Unit

Unsigned Multiply/Divide Unit
```
4820h Dividend, Bit0-7 / Multiplicand, Bit0-7
4821h Dividend, Bit8-15 / Multiplicand, Bit8-15
4822h Dividend, Bit16-23
```

```
4824h Multiplier, Bit0-7
4825h Multiplier, Bit8-15, Start Multiply on write to this register
4826h Divisor, Bit0-7
4827h Divisor, Bit8-15, Start Division on write to this register
4828h Multiply/Divide Result, Bit0-7
4829h Multiply/Divide Result, Bit8-15
482Ah Multiply/Divide Result, Bit16-23
482Bh Multiply/Divide Result, Bit24-31
482Ch Divide Remainder, Bit0-7
482Dh Divide Remainder, Bit8-15
482Eh Multiply/Divide Reset  (write = reset 4820h..482Dh) (write 00h)
482Fh Multiply/Divide Status (bit7: 0=Ready, 1=Busy)
```

Unknown Stuff
Multiply/Divide execution time is unknown. Is it constant/faster for small values? Behaviour on Divide by 0 is unknown?
Purpose of 482Eh is unknown, does it really "reset" 4820h..482Dh? Meaning that those registers are set to zero? Is that required/optional?
Are there other modes, like support for signed-numbers, or a fast 8bit*8bit multiply mode or such?

Reportedly

```
482Eh.bit0  (0=unsigned, 1=signed)
(un)signed div0 returns --> result=00000000h, remainder=dividend AND FFFFh
-80000000h/-1 returns <unknown> ?
```

SNES Cart SPC7110 with RTC-4513 Real Time Clock (1 game)

RTC from Epson/Seiko. Used by one game from Hudson Soft:

```
Far East of Eden Zero (with RTC-4513) (1995) Red Company/Hudson Soft (JP)
```

SPC7110 I/O Ports for RTC-4513 Access

```
4840h RTC Chip Select (bit0: 0=Deselect: CE=LOW, 1=Select: CE=HIGH)
4841h RTC Data Port   (bit0-3: Command/Index/Data)
4842h RTC Status      (bit7: 1=Ready, 0=Busy) (for 4bit transfers)
```

Usage
Switch CE from LOW to HIGH, send Command (03h=Write, 0Ch=Read), send starting Index (00h..0Fh), then read or write one or more 4bit Data units (index will automatically increment after each access, and wraps from 0Fh to 00h at end of data stream). Finally, switch CE back LOW.

Epson RTC-4513 Commands

```
03h    Write-Mode
0Ch    Read-Mode
```

Epson RTC-4513 Register Table

```
Index  Bit3   Bit2   Bit1    Bit0   Expl.
0      Sec3   Sec2   Sec1    Sec0   Seconds, Low
1      LOST   Sec6   Sec5    Sec4   Seconds, High
2      Min3   Min2   Min1    Min0   Minutes, Low
3      WRAP   Min6   Min5    Min4   Minutes, High
4      Hour3  Hour2  Hour1   Hour0  Hours, Low
5      WRAP   PM/AM  Hour5   Hour4  Hours, High
6      Day3   Day2   Day0    Day0   Day, Low     ;\
7      WRAP   RAM    Day5    Day4   Day, High    ;
8      Mon3   Mon2   Mon1    Mon0   Month, Low   ; or optionally,
9      WRAP   RAM    RAM     Mon4   Month, High  ; 6x4bit User RAM
A      Year3  Year2  Year1   Year0  Year, Low    ;
B      Year7  Year6  Year5   Year4  Year, High   ;/
C      WRAP   Week2  Week1   Week0  Day of Week
D      30ADJ  IRQ-F  CAL/HW  HOLD   Control Register D
E      RATE1  RATE0  DUTY    MASK   Control Register E
F      TEST   24/12  STOP    RESET  Control Register F
```

Whereas, the meaning of the various bits is:

```
Sec    Seconds (BCD, 00h..59h)
Min    Minutes (BCD, 00h..59h)
Hour   Hours   (BCD, 00h..23h or 01h..12h)
Day    Day     (BCD, 01h..31h)
Month  Month   (BCD, 01h..12h)
Year   Year    (BCD, 00h..99h)
Week   Day of Week (0..6) (Epson suggests 0=Monday as an example)
PM/AM  Set for PM, cleared for AM (is that also in 24-hour mode?)
WRAP   Time changed during access (reset on CE=LOW, set on seconds increase)
HOLD   Pause clock when set (upon clearing increase seconds by 1 if needed)
LOST   Time lost (eg. battery failure) (can be reset by writing 0)
IRQ-F  Interrupt Flag (Read-only, set when: See Rate, cleared when: See Duty)
RATE   Interrupt Rate (0=Per 1/64s, 1=Per Second, 2=Per Minute, 3=Per Hour)
DUTY   Interrupt Duty (0=7.8ms, 1=Until acknowledge, ie. until IRQ-F read)
MASK   Interrupt Disable (when set: IRQ-F always 0, STD.P always High-Z)
TEST   Reserved for Epson's use (should be 0) (auto-cleared on CE=LOW)
RAM    General purpose RAM (usually 3bits) (24bits when Calendar=off)
CAL/HW Calendar Enable (1=Yes/Normal, 0=Use Day/Mon/Year as 24bit user RAM)
24/12  24-Hour Mode (0=12, 1=24) (Time/Date may get corrupted when changed!)
30ADJ  Set seconds to zero, and, if seconds was>=30, increase minutes
STOP   Stop clock while set (0=Stop, 1=Normal)
RESET  Stop clock and reset seconds to 00h (auto-cleared when CE=LOW)
```

If WRAP=1 then one must deselect the chip, and read time/date again.

Serial data is transferred LSB first.
On-chip 32.768kHz quartz crystal.

Pin-Outs
SNES Pinouts RTC Chips

SNES Cart SPC7110 Decompression Algorithm

decompress_mode0(src,dst,len)
```
initialize
while len>0
  decoded=0
  con=0, decompression_core
  con=1+decoded, decompression_core
  con=3+decoded, decompression_core
  con=7+decoded, decompression_core
  out = (out SHL 4) XOR (((out SHR 12) XOR decoded) AND Fh)
  decoded=0
  con=15, decompression_core
  con=15+1+decoded, decompression_core
  con=15+3+decoded, decompression_core
  con=15+7+decoded, decompression_core
  out = (out SHL 4) XOR (((out SHR 12) XOR decoded) AND Fh)
  [dst]=(out AND FFh), dst=dst+1, len=len-1
```

decompress_mode1(src,dst,len)
```
initialize
while len>0
 if (buf_index AND 01h)=0
  for pixel=0 to 7
    a = (out SHR 2)  AND 03h
    b = (out SHR 14) AND 03h
    decoded=0
    con = get_con(a,b,c)
    decompression_core
    con = con*2+5+decoded
    decompression_core
    do_pixel_order(a,b,c,2,decoded)
  plane0.bits(7..0) = out.bits(15,13,11,9,7,5,3,1)
  plane1.bits(7..0) = out.bits(14,12,10,8,6,4,2,0)
  [dst]=plane0
 else
  [dst]=plane1
 buf_index=buf_index+1, dst=dst+1, len=len-1
```

decompress_mode2(src,dst,len)
```
initialize
while len>0
 if (buf_index AND 11h)=0
  for pixel=0 to 7
    a = (out SHR 0)  AND 0Fh
    b = (out SHR 28) AND 0Fh
    decoded=0
    con=0
    decompression_core
    con=decoded+1
    decompression_core
    if con=2 then con=decoded+11 else con = get_con(a,b,c)+3+decoded*5
    decompression_core
    con=Mode2ContextTable[con]+(decoded AND 1)
    decompression_core
    do_pixel_order(a,b,c,4,decoded)
  plane0.bits(7..0) = out.bits(31,27,23,19,15,11,7,3)
  plane1.bits(7..0) = out.bits(30,26,22,18,14,10,6,2)
  plane2.bits(7..0) = out.bits(29,25,21,17,13, 9,5,1)
  plane3.bits(7..0) = out.bits(28,24,20,16,12, 8,4,0)
  bitplanebuffer[buf_index+0] = plane2
  bitplanebuffer[buf_index+1] = plane3
  [dst]=plane0
 else if (buf_index AND 10h)=0
  [dst]=plane1
 else
  [dst]=bitplanebuffer[buf_index AND 0Fh]
 buf_index=buf_index+1, dst=dst+1, len=len-1
```

initialize
```
src=directory_base+(directory_index*4)
mode=[src+0]
src=[src+3]+[src+2]*100h+[src+1]*10000h  ;big-endian (!)
buf_index=0
out=00000000h
c=0
top=255
val.msb=[src], val.lsb=00h, src=src+1, in_count=0
for i=0 to 15 do pixelorder[i]=i
for i=0 to 31 do ContextIndex[i]=0, ContextInvert[i]=0
```

decompression_core
```
decoded=(decoded SHL 1) xor ContextInvert[con]
evl=ContextIndex[con]
top = top - EvolutionProb[evl]
if val.msb > top
  val.msb = val.msb-(top-1)
  top = EvolutionProb[evl]-1
  if top>79 then ContextInvert[con] = ContextInvert[con] XOR 1
  decoded = decoded xor 1
  ContextIndex[con] = EvolutionNextLps[evl]
else
  if top<=126 then ContextIndex[con] = EvolutionNextMps[evl]
while(top<=126)
  if in_count=0 then val.lsb=[src], src=src+1, in_count=8
  top = (top SHL 1)+1
  val = (val SHL 1), in_count=in_count-1    ;16bit val.msb/lsb
```

do_pixel_order(a,b,c,shift,decoded)
```
m=0, x=a, repeat, exchange(x,pixelorder[m]), m=m+1, until x=a
for m=0 to (1 shl shift)-1 do realorder[m]=pixelorder[m]
```

```
m=0, x=c, repeat, exchange(x,realorder[m]), m=m+1, until x=c
m=0, x=b, repeat, exchange(x,realorder[m]), m=m+1, until x=b
m=0, x=a, repeat, exchange(x,realorder[m]), m=m+1, until x=a
out = (out SHL shift) + realorder[decoded]
c = b
```

get_con(a,b,c)

```
if (a=b AND b=c) then return=0
else if (a=b) then return=1
else if (b=c) then return=2
else if (a=c) then return=3
else return=4
```

EvolutionProb[0..52]

```
90,37,17, 8, 3, 1,90,63,44,32,23,17,12, 9, 7, 5, 4, 3, 2
90,72,58,46,38,31,25,21,17,14,11, 9, 8, 7, 5, 4, 4, 3, 2
2 ,88,77,67,59,52,46,41,37,86,79,71,65,60,55
```

EvolutionNextLps[0..52]

```
1 , 6, 8,10,12,15, 7,19,21,22,23,25,26,28,29,31,32,34,35
20,39,40,42,44,45,46,25,26,26,27,28,29,30,31,33,33,34,35
36,39,47,48,49,50,51,44,45,47,47,48,49,50,51
```

EvolutionNextMps[0..52]

```
1 , 2, 3, 4, 5, 5, 7, 8, 9,10,11,12,13,14,15,16,17,18, 5
20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38
5 ,40,41,42,43,44,45,46,24,48,49,50,51,52,43
```

Mode2ContextTable[0..14] ;only entries 3..14 used (entries 0..2 = dummies)

```
0 ,0 ,0 ,15,17,19,21,23,25,25,25,25,25,27,29
```

SNES Cart SPC7110 Notes

Compression/Decompression Example

Uncompressed Data (64-byte ASCII string):

```
Test123.ABCDABCDAAAAAAAAaaaabbbbccccdddd7654321076543210.Test123
```

Compressed in Mode0:

```
68 91 36 15 F8 BF 42 35 2F 67 3D B7 AA 05 B4 F7 70 7A 26 20 EA 58 2C 09 61 00
C5 00 8C 6F FF D1 42 9D EE 7F 72 87 DF D6 5F 92 65 00 00
```

Compressed in Mode1:

```
4B F6 80 1E 3A 4C 42 6C DA 16 0F C6 44 ED 64 10 77 AF 50 00 05 C0 01 27 22 B0
83 51 05 32 4A 1E 74 93 08 76 07 E5 32 12 B4 99 9E 55 A3 F8 00
```

Compressed in Mode2:

```
13 B3 27 A6 F4 5C D8 ED 6C 6D F8 76 80 A7 87 20 39 4B 37 1A CC 3F E4 3D BE 65
2D 89 7E 0B 0A D3 46 D5 0C 1F D3 81 F3 AD DD E8 5C C0 BD 62 AA CB F8 B5 38 00
```

Selftest Program

All three SPC7110 games include a selftest function (which executes on initial power-up, ie. when the battery-backed SRAM is still uninitialized). Press Button A/B to start 1st/2nd test, and push Reset Button after each test.

PCBs

```
SHVC-BDH3B-01 (without RTC)
SHVC-LDH3C-01 (with RTC)
```

SNES Cart Unlicensed Variants

Gamars Puzzle (Kaiser)

A LoROM game with SRAM at 316000h (unlike normal LoROM games that have SRAM at 70xxxxh). Cartridge Header Maker entry is [FFDAh]=00h, and SRAM size entry is [FFD8h]=20h (4096 gigabytes), the actual size of the SRAM is unknown.

Bootlegs

The "bootleg" games are semi-illegal pirate productions, typically consisting of a custom (and not-so-professional) game engine, bundled with graphics and sounds ripped from commercial games. Some of these cartridges are containing some small copy-protection hardware (see below).

Copy-Protected Bootlegs (Standard "bitswap" variant)

This type is used by several games:

```
A Bug's Life                          2MB, CRC32=014F0FCFh
Aladdin 2000                          2MB, CRC32=752A25D3h
Bananas de Pijamas                    1MB, CRC32=52B0D84Bh
Digimon Adventure                     2MB, CRC32=4F660972h
King of Fighters 2000 (aka KOF2000)   3MB, CRC32=A7813943h
Pocket Monster (aka Picachu)          2MB, CRC32=892C6765h
Pokemon Gold Silver                   2MB, CRC32=7C0B798Dh
Pokemon Stadium                       2MB, CRC32=F863C642h
Soul Edge Vs Samurai                  2MB, CRC32=5E4ADA04h
Street Fighter EX Plus Alpha          2MB, CRC32=DAD59B9Fh
X-Men vs. Street Fighter              2MB, CRC32=40242231h
```

The protection hardware is mapped to:

```
80-xx:8000-FFFF  Read 8bit Latch (bits re-ordered as: 0,6,7,1,2,3,4,5)
88-xx:8000-FFFF  Write 8bit Latch (bits ordered as:   7,6,5,4,3,2,1,0)
```

Copy-Protected Bootlegs (Soulblade "constant" variant)

This type is used by only one game:

```
Soul Blade                            3MB, CRC32=C97D1D7Bh
```

The protection hardware consists of a read-only pattern, mapped to:

```
80-BF:8000-FFFF  Filled with a constant 4-byte pattern (55h,0Fh,AAh,F0h)
C0-FF:0000-FFFF  Open bus (not used)
```

Copy-Protected Bootlegs (Tekken2 "alu/flipflop" variant)

This type is used by only one game:

```
  Tekken 2                          2MB, CRC32=066687CAh
```
The protection hardware is mapped to:
```
  [80-BF:80xx]=0Fh,00h Clear all 6 bits
  [80-BF:81xx]=xxh     Probably "No Change" (unused, except for Reading)
  [80-BF:82xx]=FFh,00h Set Data bit0
  [80-BF:83xx]=FFh,00h Set Data bit1
  [80-BF:84xx]=FFh,00h Set Data bit2
  [80-BF:85xx]=FFh,00h Set Data bit3
  [80-BF:86xx]=FFh,00h Set ALU Direction bit (0=Up/Left, 1=Down/Right)
  [80-BF:87xx]=FFh,00h Set ALU Function bit  (0=Count, 1=Shift)
  X=[80-BF:81xx]       Return "4bitData plus/minus/shl/shr 1"
  ;the above specs are based on 12 known/guessed results (as guessed by d4s),
  ;the remaining 52 combinations are probably following same rules (not tested
  ;on real hardware). theoretically some ports might do things like "set bitX
  ;and clear bitY", in that case, there would be more than 64 combinations.
```
The hardware is often missing I/O accesses, unless one is repeating them some dozens of times; the existing game is issuing 240 words (480 bytes) to write-ports, and reads 256 words (512 bytes) from the read-port. The reads contain the result in lower 4bit (probably in both low-byte and high-byte of the words) (and unknown/unused stuff in the other bits).
The set/clear ports are said to react on both reads and writes (which would imply that the written data is don't care).

SNES Cart S-RTC (Realtime Clock) (1 game)

PCB "SHVC-LJ3R-01" with 24pin "Sharp S-RTC" chip. Used only by one japanese game:
```
  Dai Kaiju Monogatari 2 (1996) Birthday/Hudson Soft (JP)
```

S-RTC I/O Ports
```
  002800h S-RTC Read  (R)
  002801h S-RTC Write (W)
```
Both registers are 4bits wide. When writing: Upper 4bit should be zero. When reading: Upper 4bit should be masked-off (they do possibly contain garbage, eg. open-bus).

S-RTC Communication
The sequence for setting, and then reading the time is:
```
  Send <0Eh,04h,0Dh,0Eh,00h,Timestamp(12 digits),0Dh> to [002801h]
  If ([002800h] AND 0F)=0Fh then read <Timestamp(13 digits)>
  If ([002800h] AND 0F)=0Fh then read <Timestamp(13 digits)>
  If ([002800h] AND 0F)=0Fh then read <Timestamp(13 digits)>
  If ([002800h] AND 0F)=0Fh then read <Timestamp(13 digits)>
  etc.
```
The exact meaning of the bytes is unknown. 0Eh/0Dh seems to invoke/terminate commands, 04h might be some configuration stuff (like setting 24-hour mode). 00h is apparently the set-time command. There might be further commands (such like setting interrupts, alarm, 12-hour mode, reading battery low & error flags, etc.). When reading, 0Fh seems to indicate sth like "time available".
The 12/13-digit "SSMMHHDDMYYY(D)" Timestamps are having the following format:
```
  Seconds.lo  (BCD, 0..9)
  Seconds.hi  (BCD, 0..5)
  Minutes.lo  (BCD, 0..9)
  Minutes.hi  (BCD, 0..5)
  Hours.lo    (BCD, 0..9)
  Hours.hi    (BCD, 0..2)
  Day.lo      (BCD, 0..9)
  Day.hi      (BCD, 0..3)
  Month       (HEX, 01h..0Ch)
  Year.lo     (BCD, 0..9)
  Year.hi     (BCD, 0..9)
  Century     (HEX, 09h..0Ah for 19xx..20xx)
```
When READING the time, there is one final extra digit (the existing software doesn't transmit that extra digit on WRITING, though maybe it's possible to do writing, too):
```
  Day of Week? (0..6) (unknown if RTC assigns sth like 0=Sunday or 0=Monday)
```

Pinouts
SNES Pinouts RTC Chips

Note
There's another game that uses a different RTC chip: A 4bit serial bus RTC-4513 (as made by Epson) connected to a SPC7110 chip.

SNES Cart Super Gameboy

The Super Gameboy (SGB) is some kind of an adaptor for monochrome handheld Gameboy games. The SGB cartridge contains a fully featured gameboy (with CPU, Video & Audio controllers), but without LCD screen and without joypad buttons.
The 4-grayshade 160x144 pixel video signal is forwarded to SNES VRAM and shown on TV Set, and in the other direction, the SNES joypad data is forwarded to SGB CPU.
Some gameboy games include additional SGB features, allowing to display a 256x224 pixel border that surrounds the 160x144 pixel screen, there are also some (rather limited) functions for colorizing the monochrome screen, plus some special Sound, OBJ, Joypad functions. Finally, the gameboy game can upload program code to the SNES and execute it.

Chipset
```
  SGB CPU - 80pin - Super Gameboy CPU/Video/Audio Chip
  ICD2-R (or ICD2-N) - 44pin - Super Gameboy SGB-to-SNES Interface Chip
```
Plus VRAM/WRAM for SGB CPU, plus SNES SGB BIOS, plus CIC chip.

SGB I/O Map (ICD2-R)
```
  6000       R  LCD Character Row and Buffer Write-Row
  6001       W  Character Buffer Read Row Select
  6002       R  16-Byte Packet Available Flag
  6003       W  Reset/Multiplayer/Speed Control
  6004-6007  W  Controller Data for Player 1-4
  6008-600E  -  Unused (Open Bus, or mirror of 600Fh on some chips)
  600F       R  Chip Version (21h or 61h)
  6800-680F  -  Unused (Open Bus)
  7000-700F  R  16-byte command packet (addr 7000..700F)
```

```
  7800       R  Character Buffer Data (320 bytes of currently selected row)
  7801-780F  R  Unused (Mirrors of 7800h, not Open Bus)
```
The ICD2 chips decodes only A0-A3,A11-A15,A22 (so above is mirrored to various addresses at xx6xxN/xx7xxN). Reading the Unused registers (and write-only ones) returns garbage. On chips with [600Fh]=61h, that garbage is:
```
  CPU Open Bus values (though, for some reason, usually with bit3=1).
```
On chips with [600Fh]=21h, that garbage is:
```
  6001h.R, 6004h-6005h.R --> mirror of 6000h.R
  6003h.R, 6006h-6007h.R --> mirror of 6002h.R
  6008h-600Eh.R          --> mirror of 600Fh.R
```
On ICD2-N chips and/or such with [600Fh]=other, that garbage is: Unknown.

SGB Port 6000h - LCD Character Row and Buffer Write-Row (R)
```
  7-3  Current Character Row on Gameboy LCD (0..11h) (11h=Last Row, or Vblank)
  2    Seems to be always zero
  1-0  Current Character Row WRITE Buffer Number (0..3)
```

SGB Port 6001h - Character Buffer Read Row Select (W)
```
  7-2  Unknown/unused      (should be zero)
  1-0  Select Character Row READ Buffer Number (0..3)
```
Selects one of the four buffer rows (for reading via Port 7800h). Only the three "old" buffers should be selected, ie. not the currently written row (which is indicated in 6000h.Bit1-0).

SGB Port 6002h - 16-Byte Packet Available Flag (R)
```
  7-1  Seems to be always zero
  0    New 16-byte Packet Available (0=None, 1=Yes)
```
When set, a 16-byte SGB command packet can be read from 7000h-700Fh; of which, reading 7000h does reset the flag in 6002h.

SGB Port 6003h - Reset/Multiplayer/Speed Control (W)
```
  7    Reset Gameboy CPU   (0=Reset, 1=Normal)
  6    Unknown/unused      (should be zero)
  5-4  num_controllers     (0,1,3=One,Two,Four)  (default 0=One Player)
  3-2  Unknown/unused      (should be zero)
  1-0  SGB CPU Speed       (0..3 = 5MHz,4MHz,3MHz,2.3MHz) (default 1=4MHz)
```
The LSBs select the SGB CPU Speed (the SNES 21MHz master clock divided by 4,5,7,9). Unknown if/how/when the SGB BIOS does use this. For the SGB, the exact master clock depends on the console (PAL or NTSC). For the SGB2 it's derived from a separate 20.9MHz oscillator.

SGB Port 6004h-6007h - Controller Data for Player 1-4 (W)
```
  7    Start     (0=Pressed, 1=Released)
  6    Select    (0=Pressed, 1=Released)
  5    Button B  (0=Pressed, 1=Released)
  4    Button A  (0=Pressed, 1=Released)
  3    Down      (0=Pressed, 1=Released)
  2    Up        (0=Pressed, 1=Released)
  1    Left      (0=Pressed, 1=Released)
  0    Right     (0=Pressed, 1=Released)
```
Used to forward SNES controller data to the gameboy Joypad inputs. Ports 6005h-6007h are used only in 2-4 player mode (which can be activated via 6003h; in practice: this can be requested by SGB games via MLT_REQ (command 11h), see SGB section in Pan Docs for details).

SGB Port 600Fh - Chip Version (R)
```
  7-0  ICD2 Chip Version
```
Seems to indicate the ICD2 Chip Version. Known values/versions are:
```
  21h = ICD2-R (without company logo on chip package)
  61h = ICD2-R (with company logo on chip package)
  ??  = ICD2-N (this one is used in SGB2)
```
The versions differ on reading unused/write-only ports (see notes in SGB I/O map).

SGB Port 7000h-700Fh - 16-byte Command Packet (R)
```
  7-0  Data
```
Reading from 7000h (but not from 7001h-700Fh) does reset the flag in 6002h
Aside from regular SGB commands, the SGB BIOS (that in the SGB CPU chip) does transfer six special packets upon Reset; these do contain gameboy cartridge header bytes 104h..14Fh (ie. Nintendo Logo, Title, ROM/RAM Size, SGB-Enable bytes, etc).

SGB Port 7800h - Character Buffer Data (R)
```
  7-0  Data (320 bytes; from Buffer Row number selected in Port 6001h)
```
This port should be used as fixed DMA source address for transferring 320 bytes (one 160x8 pixel character row) to WRAM (and, once when the SNES is in Vblank, the whole 160x144 pixels can be DMAed from WRAM to VRAM).
The ICD2 chip does automatically re-arrange the pixel color signals (LD0/LD1) back to 8x8 pixel tiles with two bit-planes (ie. to the same format as used in Gameboy and SNES VRAM).
The buffer index (0..511) is reset to 0 upon writing to Port 6001h, and is automatically incremented on reading 7800h. When reading more than 320 bytes, indices 320..511 return FFh bytes (black pixels), and, after 512 bytes, it wraps to index 0 within the same buffer row.

Gameboy Audio
The stereo Gameboy Audio Output is fed to the External Audio Input on SNES cartridge port, so sound is automatically forwarded to the TV Set, ie. software doesn't need to process sound data (however, mind that the /MUTE signal of the SNES APU must be released).

SGB Commands
Above describes only the SNES side of the Super Gameboy. For the Gameboy side (ie. for info on sending SGB packets, etc), see SGB section in Pan Docs:
```
  http://nocash.emubase.de/pandocs.htm
  http://nocash.emubase.de/pandocs.txt
```
Some details that aren't described in (current) Pan Docs:
```
 * JUMP does always destroy the NMI vector (even if it's 000000h)
 * (The SGB BIOS doesn't seem to use NMIs, so destroying it doesn't harm)
 * JUMP can return via 16bit retadr (but needs to force program bank 00h)
 * After JUMP, all RAM can be used, except [0000BBh..0000BDh] (=NMI vector)
 * The IRQ/COP/BRK vectors/handlers are in ROM, ie. only NMIs can be hooked
 * APU Boot-ROM can be executed via MOV [2140h],FEh (but Echo-Write is kept on)
 * The TEST_EN command points to a RET opcode (ie. it isn't implemented)
 * Upon RESET, six packets with gameboy cart header are sent by gameboy bios
 * command 19h does allow to change an undoc flag (maybe palette related?)
 * command 1Ah..1Fh point to RET (no function) (except 1Eh = boot info)
 * sgb cpu speed can be changed (unknown if/how supported by sgb bios)
```

Note

There is a special controller, the SGB Commander (from Hori), which does reportedly have special buttons for changing the CPU speed - unknown how it is doing that (ie. unknown what data and/or ID bits it is transferring to the SNES controller port).
Probably done by sending button sequences (works also with normal joypad):

```
Codes for Super GameBoy Hardware
Enter these codes very quickly for the desired effect.
 After choosing a border from 4 - 10, press L + R to exit.
  Press L, L, L, L, R, L, L, L, L, R. - Screen Savers
 At the Super Game Boy,
  press L, L, L, R, R, R, L, L, L, R, R, R, R, R, R, R - Super Gameboy Credits
 Hold UP as you turn on the SNES and then press L, R, R, L, L, R - Toggle Speed
 During a game, press L, R, R, L, L, R - Toggle Speed
 During a game, press R, L, L, R, R, L - Toggle Sound
 --
```

Screen Savers --> Choose a border from 4 to 10 and press L + R to exit. Press L(4), R, L(4), R.
Super Gameboy Credits --> When you see the Super Game Boy screen appear, press L, L, L, R, R, R, L, L, L, R, R, R, R, R, R, R
Toggle Speed (Fast, Normal, Slow, Very Slow) Hold Up when powering up the SNES, then press L, R, R, L, L, R very fast.
Toggle Speed (Normal, Slow, Very Slow) During Gameplay, press L, R, R, L, L, R very fast.
Un/Mute Sound --> During Gameplay, press R, L, L, R, R, L quite fast.

SNES Cart Satellaview (satellite receiver & mini flashcard)

Satellaview I/O Ports
SNES Cart Satellaview I/O Map
SNES Cart Satellaview I/O Ports of MCC Memory Controller
SNES Cart Satellaview I/O Receiver Data Streams
SNES Cart Satellaview I/O Receiver Data Streams (Notes)
SNES Cart Satellaview I/O Receiver Control
SNES Cart Satellaview I/O FLASH Detection (Type 1,2,3,4)
SNES Cart Satellaview I/O FLASH Access (Type 1,3,4)
SNES Cart Satellaview I/O FLASH Access (Type 2)

Satellaview Transmission Format
SNES Cart Satellaview Packet Headers and Frames
SNES Cart Satellaview Channels and Channel Map
SNES Cart Satellaview Town Status Packet
SNES Cart Satellaview Directory Packet
SNES Cart Satellaview Expansion Data (at end of Directory Packets)
SNES Cart Satellaview Other Packets
SNES Cart Satellaview Buildings
SNES Cart Satellaview People
SNES Cart Satellaview Items

Satellaview Memory
SNES Cart Satellaview SRAM (Battery-backed)
SNES Cart Satellaview FLASH File Header
SNES Cart Satellaview BIOS Function Summary
SNES Cart Satellaview Interpreter Token Summary

Other Satellaview Info
SNES Cart Satellaview Chipsets
SNES Pinouts BSX Connectors

SNES Cart Satellaview I/O Map

Receiver I/O Map (DCD-BSA chip)

```
2188h Stream 1 Hardware Channel Number, Lsb (R/W)
2189h Stream 1 Hardware Channel Number, Msb (R/W)
218Ah Stream 1 Queue Size (number of received 1+22 byte Units) (R)
218Bh Stream 1 Queue 1-byte Status Units (Read=Data, Write=Reset)
218Ch Stream 1 Queue 22-byte Data Units  (Read=Data, Write=Reset/Ack)
218Dh Stream 1 Status Summary (R)
218Eh Stream 2 Hardware Channel Number, Lsb (R/W)
218Fh Stream 2 Hardware Channel Number, Msb (R/W)
2190h Stream 2 Queue Size (number of received 1+22 byte Units) (R)
2191h Stream 2 Queue 1-byte Status Unit(s?) (Read=Data, Write=Reset)
2192h Stream 2 Queue 22-byte? Data Unit(s?) (Read=Data, Write=Reset/Ack)
2193h Stream 2 Status Summary (R)
2194h POWER (bit0) and ACCESS (bit2-3) LED Control? (R/W)
2195h Unknown/Unused, maybe for EXT Expansion Port (?)
2196h Status (only bit1 is tested) (R)
2197h Control (only bit7 is modified) (R/W)
2198h Serial I/O Port 1 (R/W)
2199h Serial I/O Port 2 (R/W)
```

Flash Card I/O Map (when mapped to bank C0h and up)

```
C00000h  Type 1-4   Detection Command             (W)
C00002h  Type 1-4   Detection Status              (R)
C0FFxxh  Type 1-4   Detection Response            (R)
C00000h  Type 1,3,4 Command for Type 1,3,4        (W)
C00000h  Type 1,3,4 Status (normal commands)      (R)
C00004h  Type 1,3   Status (erase-entire command) (R)
C02AAAh  Type 2     Command/Key for Type2         (W)
C05555h  Type 2     Command/Status for Type2      (R/W)
xx0000h  Type 1-4   Erase 64K Sector Address      (W)
xxxxxxh  Type 1-4   Write Data Address            (W)
```

BIOS Cartridge MCC-BSC Chip Ports

```
005000h Unknown/Unused
015000h Bank 00h-3Fh and 80h-FFh (0=FLASH, 1=PSRAM) (?)
```

```
  025000h Mapping for PSRAM/FLASH (0=32K/LoROM, 1=64K/HiROM)
  035000h Bank 60h-6Fh (0=FLASH, 1=PSRAM) (?)
  045000h Unknown (set when mapping PSRAM as Executable or Streaming Buffer)
  055000h Bank 40h-4Fh (0=PSRAM, 1=FLASH) ;\probably also affects Banks 00h-3Fh
  065000h Bank 50h-5Fh (0=PSRAM, 1=FLASH) ;/and maybe 80h-BFh when BIOS is off?
  075000h Bank 00h-1Fh (0=PSRAM/FLASH, 1=BIOS)
  085000h Bank 80h-9Fh (0=PSRAM/FLASH, 1=BIOS)
  095000h Unknown/Unused (except: used by BS Dragon Quest, set to 00h)
  0A5000h Unknown/Unused (except: used by BS Dragon Quest, set to 80h)
  0B5000h Unknown/Unused (except: used by BS Dragon Quest, set to 80h)
  0C5000h Bank C0h-FFh FLASH Reads? (0=Disable, 1=Enable)
  0D5000h Bank C0h-FFh FLASH Writes (0=Disable, 1=Enable)
  0E5000h Apply Changes to Other MCC Registers (0=Unused/Reserved, 1=Apply)
  0F5000h Unknown/Unused
```

Bits C and D are R/W (the other ones maybe, too).

SNES Cart Satellaview I/O Ports of MCC Memory Controller

MCC I/O Ports

The MCC chip is a simple 16bit register, with the bits scattered across various memory banks (probably because the MCC chip doesn't have enough pins to decode lower address bits).

To change a bit: [bit_number*10000h+5000h]=bit_value*80h

```
  005000h Unknown/Unused
  015000h Bank 00h-3Fh and 80h-FFh (0=FLASH, 1=PSRAM) (?)
  025000h Mapping for PSRAM/FLASH (0=32K/LoROM, 1=64K/HiROM)
  035000h Bank 60h-6Fh (0=FLASH, 1=PSRAM) (?)
  045000h Unknown (set when mapping PSRAM as Executable or Streaming Buffer)
  055000h Bank 40h-4Fh (0=PSRAM, 1=FLASH) ;\probably also affects Banks 00h-3Fh
  065000h Bank 50h-5Fh (0=PSRAM, 1=FLASH) ;/and maybe 80h-BFh when BIOS is off?
  075000h Bank 00h-1Fh (0=PSRAM/FLASH, 1=BIOS)
  085000h Bank 80h-9Fh (0=PSRAM/FLASH, 1=BIOS)
  095000h Unknown/Unused
  0A5000h Unknown/Unused
  0B5000h Unknown/Unused
  0C5000h Bank C0h-FFh FLASH Reads? (0=Disable, 1=Enable)
  0D5000h Bank C0h-FFh FLASH Writes (0=Disable, 1=Enable)
  0E5000h Apply Changes to Other MCC Registers (0=Unused/Reserved, 1=Apply)
  0F5000h Unknown/Unused
```

Bits C and D are R/W (the other ones maybe, too, probably except bit E)
Bit 5,6 might also enable FLASH reads,writes in bank 40h-7Dh ?

Satellaview BIOS Cartridge Memory Map

```
  00-0F:5000       MCC I/O Ports (Memory Control, BIOS/PSRAM/FLASH Enable)
  10-1F:5000-5FFF  SRAM            (32Kbyte SRAM in 4K-banks)
  xx-3F:6000-7FFF  PSRAM        (Mirror of 8K at PSRAM offset 06000h..07FFFh)
  00-3F:8000-FFFF  PSRAM/FLASH/BIOS in 32K-banks (Slow LoROM mapping)
  40-4F:0000-FFFF  PSRAM/FLASH  (for Executables with Slow HiROM mapping)
  50-5F:0000-FFFF  PSRAM/FLASH  (for Executables with Slow HiROM mapping)
  60-6F:0000-FFFF  FLASH/PSRAM  (for use as Work RAM or Data Files)
  70-77:0000-FFFF  PSRAM
  80-BF:8000-FFFF  PSRAM/FLASH/BIOS  in 32K-banks (Fast LoROM mapping)
  C0-FF:0000-FFFF  PSRAM/FLASH       (FLASH with R/W Access)
```

Memory

```
  BIOS ROM  1MByte (LoROM mapping, 20h banks of 32Kbytes each)
  FLASH     1Mbyte   (can be mapped as LoROM, HiROM, or Work Storage)
  PSRAM     512Kbyte (can be mapped as LoROM, HiROM, or Work RAM)
  SRAM      32Kbyte (mapped in eight 4K banks)
```

Note: FLASH is on an external cartridge, size is usually 1MByte (as shown above).

SNES Cart Satellaview I/O Receiver Data Streams

The receiver can be programmed to watch (and receive) two different Hardware Channels simultaneously. In practice, Stream 1 is used only by the BIOS, and Stream 2 is used only by a few BS FLASH games (Dragon Quest 1, Satella2 1, BS Fire Emblem Akaneia Senki 1, and maybe some others) (which do use it for receiving Time Channel Packets).

2188h/2189h Stream 1 Hardware Channel Number, Lsb/Msb (R/W)
218Eh/218Fh Stream 2 Hardware Channel Number, Lsb/Msb (R/W)

```
  0-15  Hardware Channel Number (16bit)
          XXX reportedly only 14bit !?
```

Values written to these registers should be taken from the Channel Map packet (or for receiving the Channel Map itself, use fixed value 0124h). Be sure to reset the Queues after changing the channel number (so you won't receive old data from old channel).

218Ah Stream 1 Queue Size (number of received 1+22 byte Units) (R)
2190h Stream 2 Queue Size (number of received 1+22 byte Units) (R)

```
  0-6  Number of received Units contained in the Queue (0..127)
  7    Overrun Error Flag (set when received more than 127 units)
```

Indicates how many frames are in the queues. One doesn't need to process all frames at once; when reading only a few frames, the Queue Size is decremented accordingly, and the remaining frames stay in the Queue so they can be processed at a later time. The decrement occurs either after reading 1 byte from the Status Queue, or after reading 22 bytes from the Data Queue (anyways, to keep the queues in sync, one should always read the same amount of 1/22-byte Units from both Queues, so it doesn't matter when the decrement occurs).

218Bh Stream 1 Queue 1-byte Status Units (Read=Data, Write=Reset)
2191h Stream 2 Queue 1-byte Status Unit(s?) (Read=Data, Write=Reset)

Contains Header/Data Start/End flags for the received Data Frames, the format seems to be same as for Port 218Dh/2193h (see there for details) (if it's really same, then the two Error bits should be also contained in the Status Queue, though the BIOS doesn't use them in that place).

218Ch Stream 1 Queue 22-byte Data Units (Read=Data, Write=Reset/Ack)
2192h Stream 2 Queue 22-byte? Data Unit(s?) (Read=Data, Write=Reset/Ack)

Contains the received Data Frames, or in case of Header Frames: The 5/10-byte Frame Header, followed by the Packet/Fragment Header, followed by the actual

Data.

218Dh Stream 1 Status Summary (R)
2193h Stream 2 Status Summary (R)
These registers seem to contain a summary of the Status bytes being most recently removed from the Queue. Ie. status bits are probably getting set by ORing all values being read from Port 218Ah/2190h. The bits are probably cleared after reading 218Dh/2193h.

```
0-1  Unknown/unused
2-3  Error Flags (probably set on checksum errors or lost data/timeouts)
4    Packet Start Flag (0=Normal, 1=First Frame of Packet) (with Header)
5-6  Unknown/unused
7    Packet End Flag   (0=Normal, 1=Last Frame of Packet)
```

Bit 2-3 are more or less self-explaining: Don't use the queued data, and discard any already (but still incompletely) received packet fragments. Bit 4,7 are a bit more complicated. See Notes in next chapter for details.
SNES Cart Satellaview I/O Receiver Data Streams (Notes)

SNES Cart Satellaview I/O Receiver Data Streams (Notes)

Resetting the Queues
Clearing the Status & Data Queues is needed on power-up, after Overrun, or after changing the Hardware Channel number. The procedure is:

```
MOV A,01h     ;\
MOV [218Bh],A ; must be executed in FAST memory (at 3.58MHz) (otherwise the
NOP           ; the Status Queue may be not in sync with the Data Queue)
NOP           ; (for Stream 2 do the same with Port 2192h/2193h accordingly,
NOP           ; though the existing games that do use Stream 2 are including
NOP           ; several near-excessive timing bugs in that section)
MOV [218Ch],A ;/
```

Thereafter, Status & Data queue are empty, and the Queue Size register is 00h (both 7bit counter, and Overrun flag cleared).

Reading the Queues

```
N=[218Ah]                                  ;-get queue size
if N=0 then exit                           ;-exit if no data in queues
if N.Bit7=1 then reset_queue/abort_packet/exit ;-handle overrun error
N=max(20,N)                                ;-limit to max 20 (if desired)
for i=0 to (N-1), stat[i]=[219Bh], next    ;-read status units
stat_summary=[219Dh]                       ;-get status summary
for i=0 to (N*22-1), data[i]=[219Ch], next ;-read data units
```

Channel Disable
After receiving a full packet, the BIOS issues a "MOV [218Ch],00h", this might acknowledge something, or (more probably) disable the Channel so that no new data is added to the Queue. The mechanism for re-enabling the channel is unknown (prossibly resetting the Queue, or writing the Channel register). For Stream 2, "MOV [2192h],00h" should do the same thing.

Overrun Notes
Overrun means that one hasn't processed the queues fast enough. If so, one should Reset the queues and discard any already-received incomplete packet fragments. There seems to be no problem if an overrun occurs WHILE reading from the queue (ie. overrun seems to stop adding data to the queue, rather than overwriting the old queued data) (of course AFTER reading the queue, one will need to handle the overrun, ie. discard all newer data).
Note: Stream 1 can queue 127 frames (presumably plus 1 incomplete frame, being currently received). As far as known, Stream 2 is used only for Time Channels (with single-frame packets), so it's unknown if Stream 2 is having the same queue size.

Packet Start/End Flags
The status queue values (with start/end bits isolated) would be:

```
90h              ;packet is 1 frame  (10-byte header + 12-byte data)
10h,80h          ;packet is 2 frames (10-byte header + 34-byte data)
10h,00h,80h      ;packet is 3 frames (10-byte header + 56-byte data)
10h,00h,00h,80h  ;packet is 4 frames (10-byte header + 78-byte data)
```

and so on. For Channel Map, header is only 5-byte, and data is 5 bigger.

Caution: After having received the header (ie. at time when receiving the data), the BIOS treats the Header-Start Flag in the Status Summary register (!) as Error-Flag, to some point that makes sense in the Data-phase, but it will cause an error if a new header frame is received shortly AFTER the Data-phase. As a workaround, the transmitter should not send new Packet Fragments (on the same Hardware Channel) for at least 1/60 seconds (preferably longer, say 1/20 seconds) after the end of the last Data Frame.

Transfer Rate
The transfer rate is unknown. Aside from the actual transmission speed, the effective download rate will also depend on how often data is transmitted on a specific channel (there are probably pauses between packet fragments, and maybe also between 22-byte frames), this may vary depening on how many other packets are transmitted, and how much priority is given to individual packets.
The download rate is slow enough for allowing the BIOS to write incoming data directly to FLASH memory. Moreover, the BIOS Vblank NMI Handler processes only max twenty 22-byte frames per 60Hz PPU frame. Knowing that, the download rate must be definetly below 26400 bytes/second (20*22*60).
As far as known, the Satellaview broadcasts replaced former St.GIGA radio broadcasts, assuming that the radio used uncompressed CD quality (2x16bit at 44.1kHz), and assuming that Satellaview used the same amount of data, the transfer rate may have been 176400 bytes/second (which could have been divided to transfer 8 different packets at 22050 bytes/second, for example).

SNES Cart Satellaview I/O Receiver Control

2194h POWER (bit0) and ACCESS (bit2-3) LED Control? (R/W)

```
0   Usually set  <-- is ZERO by Itoi (maybe POWER LED) (see? 2196h.Bit0)
1   Usually zero <-- is SET by Itoi                    (see? 2196h.Bit0)
2-3 Usually both set or both cleared (maybe ACCESS LED) (Bit2 is Access LED)
4-7 Usually zero
```

Bit2/3 are toggled by software when writing to FLASH memory. Bit0 is usually set. Might control the POWER and ACCESS LEDs on the Satellaview's Front Panel (assuming that the LEDs are software controlled). Using other values than listed above might change the LED color (assuming they are two-color LEDs).

2195h Unknown/Unused, maybe for EXT Expansion Port (?)
This register isn't used by the BIOS, nor by any games. Maybe it does allow to input/output data to the Satellaview's EXT Port.

2196h Status (only bit1 is tested) (R)

```
0   Unknown (reportedly toggles at fast speed when 2194h.Bit0-or-1? is set)
```

```
  1    Status (0=Okay, 1=Malfunction)
  2-7  Unknown/unused
```
The BIOS is using only Bit1, that bit is tested shortly after the overall hardware detection, and also during NMI handling. Probably indicates some kind of fundamental problem (like low supply voltage, missing EXPAND-Pin connection in cartridge, or no Satellite Tuner connected).

2197h Control (only bit7 is modified) (R/W)
```
  0-6  Unknown/unused (should be left unchanged)
  7    Power Down Mode? (0=Power Down, 1=Operate/Normal) (Soundlink enable?)
```
Bit7 is set by various BIOS functions, and, notably: When [7FD9h/FFD9h].Bit4 (in Satellaview FLASH File Header) is set. Also notably: Bit7 is set/cleared depending on Town Status Entry[07h].Bit6-7.

2198h Serial I/O Port 1 (R/W)
2199h Serial I/O Port 2 (R/W)
These ports are basically 3-bit parallel ports, which can be used as three-wire serial ports (with clock, data.in, data.out lines) (by doing the "serial" transfer by software). Outgoing data must be written before toggling clock, incoming data can be read thereafter.
```
  0    Clock (must be manually toggled per data bit)
  1-5  Unknown/unused (should be 0)
  6    Chip Select - For Port 1: 1=Select / For Port 2: 0=Select
  7    Data (Write=Data.Out, Read=Data.in) (data-in is directly poll-able)
```
Bits are transferred MSB first.
Unknown which chips these ports are connected to. One port does most probably connect to the 64pin MN88821 chip (which should do have a serial port; assuming that it is a MN88831 variant). The other port <might> connect to the small 8pin SPR-BSA chip?
Possible purposes might be configuration/calibration, Audio volume control, and Audio channel selection (assuming that the hardware can decode audio data and inject it to SNES Expansion Port sound inputs).

Serial Port 1 (2198h)
The BIOS contains several functions for sending multi-byte data to, and receiving 16bit-units from this port. Though the functions seem to be left unused? (at least, they aren't used in the low-level portion in first 32K of the BIOS).
Port 1 specific notes: When reading (without sending), the outgoing dummy-bits should be set to all zero. Chip is selected when Bit6=1. Aside from receiving data from bit7, that bit is also polled in some cases for sensing if the chip is ready (0=Busy, 1=Ready).

Serial Port 2 (2199h)
Data written to this port consists of simple 2-byte pairs (index-byte, data-byte), apparently to configure some 8bit registers. Used values are:
```
  Reg[0] = 88h (or 00h when Power-Down?) (soundlink on/off?)
  Reg[1] = 80h
  Reg[2] = 04h
  Reg[3] = 00h
  Reg[4] = 08h
  Reg[5] = 00h
  Reg[6] = 70h
  Reg[7] = Not used
  Reg[8] = 00h
  Reg[9..FF] = Not used
```
There are also BIOS functions for reading 1-byte or 3-bytes from this Port, but they seem to be left unused (but, BS Dragon Quest, and Itoi are doing 24bit reads via direct I/O, whereas Itoi wants the 1st bit to be 0=ready/okay).
Port 2 specific notes: When reading (without sending), the outgoing dummy-bits should be set to all ones. Chip(-writing) is selected when Bit6=0.

SNES Cart Satellaview I/O FLASH Detection (Type 1,2,3,4)

The Satellaview FLASH cartridges contain slightly customized "standard" FLASH chips; with a Nintendo-specific Chip Detection sequence:

Detection Sequence
```
  [C00000h]=38h, [C00000h]=D0h                  ;request chip info part 1
  delay (push/pop A, three times each)          ;delay
  [C00000h]=71h                                 ;enter status mode
  repeat, X=[C00002h], until (X.bit7=1)         ;wait until ready
  [C00000h]=72h, [C00000h]=75h                  ;request chip info part 2
  FOR i=0 to 9, info[i]=BYTE[C0FF00h+i*2], NEXT ;read chip info (10 bytes)
  [C00000h]=FFh                                 ;terminate status mode
```
BUG: For Type 2 chips, one <should> probably use "[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h" instead of "[C00000h]=FFh" (the BIOS is actually <trying> to do that, but it's doing that before it has deciphered the Type bits).

Detection Values
```
  info[0] - ID1 (Must be "M" aka 4Dh)
  info[1] - ID2 (Must be "P" aka 50h)
  info[2] - Flags (Must be bit7=0 and bit0=0) (other bits unknown)
  info[3] - Device Info (upper 4bit=Type, lower 4bit=Size)
  info[4..9] - Unknown/Unused (BIOS copies them to RAM, but doesn't use them)
```
Type must be 01h..04h for Type 1-4 accordingly. Size must be 07h..0Ch for 128Kbyte, 256Kbyte, 512Kbyte, 1Mbyte, 2Mbyte, 4Mbyte accordingly (ie. 1 SHL N Kbytes).

Rejected Values
Wrong ID1/ID2 or wrong Flag Bit7/Bit0 are rejected. Type 00h or 05h..0Fh are rejected. Size 00h..05h is rejected. Size 06h would be a half 128Kbyte block, which is rounded-down to 0 blocks by the BIOS. Size 0Dh would exceed the 32bit block allocation flags in header entry 7FD0h/FFD0h. Size 0Fh would additionally exceed the 8bit size number.

Special Cases
If no FLASH cartridge is inserted, then the detection does probably rely on open bus values, ie. "MOV A,[C00002h]" probably needs to return C0h (the last opcode byte) which has bit7=1, otherwise the detection wait-loop would hang forever.
There are reportedly some "write-protected" cartridges. Unknown what that means, ROM-cartridges, or FLASH-cartridges with some (or ALL) sectors being write-protected. And unknown what detection values they do return.

FLASH Base Address
The Satellaview BIOS always uses C00000h as Base Address when writing commands to FLASH, the MCC chip could be programmed to mirror FLASH to other locations (although unknown if they are write-able, if so, commands could be also written to that mirrors).
Game Cartridges with built-in FLASH cartridge slot may map FLASH to other locations than C00000h, details on that games are unknown. The game carts don't include MCC chips, but other mapping hardware: In at least some of them the mapping seems to be controlled by a SA-1 chip (an external 10.74MHz 65C816 CPU with on-chip I/O ports, including memory-mapping facilities), the SA-1 doesn't seem to have FLASH-specific mapping registers, so the FLASH might be mapped as secondary ROM-chip. There may be also other games without SA-1, using different FLASH mapping mechanism(s)? For some details, see:

SNES Cart Data Pack Slots (satellaview-like mini-cartridge slot)

General Notes
FLASH erase sets all bytes to FFh. FLASH writes can only change bits from 1 to 0. Thus, one must normally erase before writing (exceptions are, for example, clearing the "Limited-Start" bits in Satellaview file header). Type 2 can write 128 bytes at once (when writing less bytes, the other bytes in that area are left unchanged). The status/detection values may be mirrored to various addresses; the normal FLASH memory may be unavailable during write/erase/detection (ie. don't try to access FLASH memory, or even to execute program code in it, during that operations).

SNES Cart Satellaview I/O FLASH Access (Type 1,3,4)

The Type 1,3,4 protocol is somewhat compatible to Sharp LH28F032SU/LH28F320SK chips (which has also a same/similar 52pin package). Concerning the commands used by the BIOS, Type 1,3,4 seems to be exactly the same - expect that Type 3 doesn't support the Erase-Entire chip command.

Erase Entire Chip (Type 1 and 4 only) (not supported by Type 3)
```
  [C00000h]=50h                             ;clear status register
  [C00000h]=71h                             ;enter status mode
  repeat, X=[C00004h], until (X.bit3=0)     ;wait until VPP voltage okay
  [C00000h]=A7h  ;"erase all unlocked pages"?  ;select erase entire-chip mode
  [C00000h]=D0h                             ;start erase
  [C00000h]=71h                             ;enter status mode
  repeat, X=[C00004h], until (X.bit7=1)     ;wait until ready
  if (X.bit5=1) then set erase error flag   ;check if erase error
  [C00000h]=FFh                             ;terminate status mode
```

Unknown Command
Same as Erase Entire (see above), but using 97h instead of A7h, and implemented ONLY for Type 1 and 4, that is: NOT supported (nor simulated by other commands) for neither Type 2 nor Type 3. Maybe A7h erases only unlocked pages, and 97h tries to erase all pages (and fails if some are locked?).

Erase 64KByte Sector
```
  [C00000h]=50h                             ;clear status register
  [C00000h]=20h                             ;select erase sector mode
  [nn0000h]=D0h                             ;start erase 64K bank nn
  [C00000h]=70h                             ;enter status mode
  repeat, X=[C00000h], until (X.bit7=1)     ;wait until ready
 ;; if (X.bit5=1) then set erase error flag    ;check if erase error
  [C00000h]=FFh                             ;terminate status mode
```

Write Data
```
  FOR i=first to last
    [C00000h]=10h                           ;write byte command
    [nnnnnnh+i]=data[i]                     ;write one data byte
    [C00000h]=70h                           ;enter status mode
    repeat, X=[C00000h], until (X.bit7=1)   ;wait until ready
  NEXT i
  [C00000h]=70h                             ;enter status mode
  repeat, X=[C00000h], until (X.bit7=1)     ;hmmm, wait again
  if (X.bit4=1) then set write error flag   ;check if write error
  [C00000h]=FFh                             ;terminate status mode
```

Enter Erase-Status Mode
```
  [C00000h]=71h                             ;enter status mode
  X=[C00004h]                               ;read status byte
  IF (X.bit7=0) THEN busy
  IF (X.bit3=1) THEN not-yet-ready-to-erase (VPP voltage low)
  IF (X.bit7=1) AND (X.bit5=0) THEN ready/okay
  IF (X.bit7=1) AND (X.bit5=1) THEN erase error ?
```

Enter Other-Status Mode
```
  [C00000h]=70h                             ;enter status mode
```

Terminate Command
```
  [C00000h]=FFh                             ;terminate
```
Used to leave status or chip-detection mode.

BUGs: On Type 3 chips, the BIOS tries to simulate the "Erase-Entire" command by issuing multiple "Erase-Sector" commands, the bug there is that it tests bit4 of the flash_size in 128Kbyte blocks (rather than bit4 of the flash_status) as erase-error flag (see 80BED2h); in practice, that means that the erase-entire will always fail on 2MByte chips (and always pass okay on all other chips); whereas, erase-entire is used when downloading files (except for small files that can be downloaded or relocated to PSRAM).

SNES Cart Satellaview I/O FLASH Access (Type 2)

Type 2 protocol is completely different as for Type 1,3,4 (aside from the Chip Detection sequence, which is same for all types).

Erase Entire Chip
```
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=80h   ;unlock erase
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=10h   ;do erase entire chip
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=70h   ;enter status mode
  repeat, X=[C05555h], until (X.bit7=1)         ;wait until ready
  if (X.bit5=1) then set erase error flag       ;check if erase error
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h   ;terminate status mode
```

Erase 64KByte Sector
```
  [C00000h]=50h                                 ;huh? (maybe a BIOS bug)
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=80h   ;unlock erase
  [C05555h]=AAh, [C02AAAh]=55h, [nn0000h]=30h   ;do erase bank nn
  repeat, X=[C05555h], until (X.bit7=1)         ;wait until ready
  if (X.bit5=1) then set erase error flag       ;check if erase error
  [C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h   ;terminate status mode
```

Write 1..128 Bytes (within a 128-byte boundary)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=A0h   ;enter write mode
FOR i=first to last, [nnnnnn+i]=data[i]       ;write 1..128 byte(s)
[nnnnnn+last]=DATA[last]   ;write LAST AGAIN  ;start write operation
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=70h   ;enter status mode
repeat, X=[C05555h], until (X.bit7=1)         ;wait until ready
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h   ;terminate status mode
```

Enter Status Mode

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=70h   ;enter status mode
X=[C05555h]                                   ;read status byte
IF (X.bit7=0) THEN busy
IF (X.bit7=1) AND (X.bit5=0) THEN ready/okay
IF (X.bit7=1) AND (X.bit5=1) THEN ready/erase error ?
```

Terminate Command

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h   ;terminate
```

Used to leave status or chip-detection mode.

Bugged Commands

The are some cases (BIOS addresses 80BF88h, 80DBA4h, 80E0D5h) where Type 2 programming is mixed up with some non-Type-2 commands (setting [C00000h]=50h and [C00000h]=FFh), these commands are probably ignored by chip (or switching it back default mode).

SNES Cart Satellaview Packet Headers and Frames

Packet Fragment Format (10-byte header)

```
00h 1    Transmission ID (in upper 4bit) (must stay same for all fragments)
01h 1    Current Fragment Number (in lower 7bit)
02h 3    Fragment Size (N) (big-endian) (excluding first 5 bytes at [00..04])
05h 1    Fixed, Must be 01h
06h 1    Total Number of Fragments (00h=Infinite Streaming?)
07h 3    Target Offset (big-endian) (location of fragment within whole file)
0Ah N-5  Data Body (first 12 bytes located directly in header frame)
... ...  Unused/Padding (until begin of next 22-byte Frame)
```

This is the normal format used by all packets (except Channel Map packet).

Channel Map Packet Format (5-byte header)

```
00h 1    Unknown/unused (would be 4bit Transmission ID for normal packets)
01h 1    Unknown/unused (would be 7bit Fragment Number for normal packets)
02h 3    Packet Size (N) (big-endian) (excluding first 5 bytes at [00..04])
05h N    Data Body (first 17 bytes located directly in header frame)
... ...  Unused/Padding (until begin of next 22-byte Frame)
```

Frames (22-bytes)

Packets are divided into one or more 22-byte frames.
For each frame, a 1-byte status info can be read from Port 218Bh/2191h, the status contains error flags (indicating bad checksums or lost-frames or so), and header flags (indicating begin/end of header/data or so).
The 22-byte frame data can be read from Port 218Ch/2192h. If it is header frame, then its first 10 (or 5) bytes contain the header (as described above), and the remaining 12 (or 17) bytes are data. If it is a data frame, then all 22 bytes are plain data.

Fragmented Packets

A packet can consist of 1..128 fragments. The packet transmission is repeated several times (say, for one hour). If it is consisting of several fragments, one can start downloading anywhere, eg. with the middle fragment, and one can keep downloading even if some fragments had transmission errors (in both cases, one must download the missing fragments in the next pass(es) when the transmission is repeated).
Software should maintain a list of fragments that are already received (if a fragment is already received: just remove it from the queue without writing to target area; both for saving CPU load, and for avoiding to destroy previously received packets in case of transmission errors).
At some time (say, after an hour), transmission of the packet will end, and a different packet may be transferred on the same hardware channel. Verify the 4bit Transmission ID to avoid mixing up fragments from the old (desired) file with the new file. Ideally, that ID <should> be stored in the Channel Map or Directory (this may actually be so), however, the Satellaview BIOS simply takes the ID from the first received Fragment, and then compares it against following Fragments (if the ID changed shortly after checking the Directory, and before receiving the first fragment, then the BIOS will download a "wrong" file).

Fragment Size Bug

The Satellaview BIOS supports only 16bit fragment sizes, it does try to handle 24bit sizes, but if the fragment size exceeds 65535 then it does receive only the first few bytes, and then treats the whole fragment as "fully" received.

Text Strings

Text strings (file/folder names and descriptions) can contain ASCII (and presumably JIS and SHIFT-JIS), and following specials:

```
00h        End of Line (or return from a "\s" Sub-String)
0Dh        Carriage Return Line Feed (in descriptions)
20h..7Eh   ASCII 6pix characters (unlike SHIFT-JIS 12pix ones)
80h..9Fh   Prefixes for double-byte characters (SHIFT-JIS)
A0h..DFh   Japanese single-byte characters (JIS or so)
E0h..EAh   Prefixes for double-byte characters (SHIFT-JIS)
F0h        Prefix for Symbols (40h..51h:Music-Note,Heart,Dots,Faces,MaKenji)
"\\"           Yen symbol (unlike ASCII, not a backslash)
"\b0".."\b3"   Insert Username/Money/Gender/NumItems (12pix SHIFT-JIS)
"\c0".."\c5"   Changes color or palette or so
"\d#",p24bit   Insert 16bit Decimal at [p24bit] using 6pix-font
"\D#",p24bit   Insert 16bit Decimal at [p24bit] using 12pix-font
"\du#",v24bit  Insert 16bit Decimal Interpreter-Variable using 6pix-font
"\Du#",v24bit  Insert 16bit Decimal Interpreter-Variable using 12pix-font
                  # = 00     Variable width (no leading spaces/zeroes)
                  # = 1..6   Width 1..6 chars (with leading spaces)
                  # = 01..06 Width 1..6 chars (with leading zeroes)
"\s",ptr24bit  Insert Sub-string (don't nest with further "\s,\d,\D")
"\g",ptr24bit  Insert Custom Graphics/Symbol (ptr to xsiz,ysiz,bitmap)
"\i"           Carriage Return (set x=0, keep y=unchanged) (not so useful)
"\m0".."\m3"   Flags (bit0=ForceHorizontal16pixGrid, bit1=DonNotUpdateBg3Yet)
"\n"           Carriage Return Line Feed (same as 0Dh)
"\p00".."\p07" Palette
"\w00".."\w99" Character Delay in Frames (00=None)
"\x00".."\xNN" Set Xloc
"\y00".."\yNN" Set Yloc
```

Note: "\m","\p","\w","\x","\y" are slightly bugged (causing stack overflows when using them too often within a single string; the text output thread quits at the string-end, which somewhat 'fixes' the stack-problem).
CRLF can be used in Item Activation Messages, Folder or Download-File Descriptions.
Multi-line messages are wrapped to the next line when reaching the end of a line (the wrapping can occur anywhere within words, to avoid that effect one must manually insert CRLF's (0Dh) at suitable locations). Some message boxes are clipped to the visible number of lines, other messages boxes prompt the user to push Button-A to read further lines.
Caution: CRLF will hang in Item-Descriptions (they do work in item shops, but will hang in the Inventory menu; the only way to implement longer descriptions here is to space-pad them so that the wrapping occurs at a suitable location).

SNES Cart Satellaview Channels and Channel Map

Channels
Transmission is organized in "channels". Each can Channel transmits a single Packet (which contains a File, or other special information like in Directory and Time packets). There can be (theoretically) up to 4 billion logical "Software Channels", but only 65536 physical "Hardware Channels". The latter ones are those being currently transmitted, and which can be received by programming the channel number into Port 2188h/218Eh.

Channel Map Packet (Hardware Channel 0124h)
Unlike normal packets (with 10-byte headers), this packet is preceeded by a 5-byte header.
Loaded to 7E9BECh. Unspecified size (size should be max 1485 bytes or less, otherwise it'd overlap the Welcome Message at 7EA1B9h) (with that size limit, the map can contain max 113 channels) (but, caution: Itoi works only with max 1007 bytes (1024-byte buffer, cropped to N*22-bytes, minus 5-byte packet header)).
```
  00h 2    ID 53h,46h ("SF")
  02h 4    Unknown/unused
  06h 1    Number of entries (must be at least 1)
  07h 1    Checksum (above 7 bytes at [00..06] added together)
  08h ..   Entries (each one is 3+N*13 bytes)
```
Each entry is: (Packet Groups)
```
  00h 2    Software Channel Number (first 2 bytes, of total 4 bytes)
  02h 1    Number of sub-entries (N) (must be at least 1)
  03h N*13 Sub-entries (each one is 13 bytes)
```
Each sub-entry is: (Separate Packets)
```
  00h 1    Unknown/unused
  01h 2    Software Channel Number (last 2 bytes, of total 4 bytes)
  03h 5    Unknown/unused
  08h 2    Fragment Interval (in seconds) (big-endian) (for use as timeout)
  0Ah 1    Type/Target (lower 4bit indicate transfer method or so)
            Bit0-1: Autostart after Download (0=No, 1=Optional, 2=Yes, 3=Crash)
            Bit2-3: Target (0=WRAM, 1=PSRAM, 2=EntireFLASH, 3=FreeFLASH)
            Bit4-7: Unknown/Unused
  0Bh 2    Hardware Channel Number (2 bytes) (for Port 2188h/218Eh)
```
The transmission timeout for the Channel Map itself is 7 seconds.

Hardware Channels (2-byte / 16bit) (XXX or only 14bit?!)
```
  0121h     Used for hardware-connection test (received data is ignored)
  0124h     Channel Map
  AAEEh     Dummy number (often used to indicate an absent Time Channel)
  NNNNh     Other Hardware Channels (as listed in Channel Map)
  [7FFFF7h] Incoming Time Channel value for some games (from separate loader?)
```

Software Channels (4-byte pairs / 32bit)
```
  1.1.0.4    Welcome Message (100 bytes)
  1.1.0.5    Town Status (256 bytes)
  1.1.0.6    Directory (16Kbytes)
  1.1.0.7    SNES Patch (16Kbytes)
  1.1.0.8    Time Channel (used by BS Satella2 1, BS Fire Emblem, and Itoi)
  1.2.0.48   Time Channel (used by Dragon Quest 1, BS Zelda no Densetsu Remix)
  ?.?.?.?    Time Channel (for BS Zelda - Kodai no Sekiban Dai 3 Hanashi)
  1.2.129.0  Special Channel used by Derby Stallion 96 <-- on roof of building
  1.2.129.16 Special Channel used by Derby Stallion 96 <-- 6th main menu option
  1.2.130.N  Special Channel(s) used by Itoi Shigesato no Bass Tsuri No. 1
  N.N.N.N    Other Software Channels (as listed in Directory)
  N.N.0.0    None (for directory entries that have no File or Include File)
```

Endianess of Numbers in Satellite Packets
In the satellite packets, all 16bit/24bit values (such like length or address offsets) are in big-endian format (opposite of the SNES CPUs byte-order). For the Hardware/Software Channel Numbers it's hard to say if they are meant to be big-endian, little-endian, or if they are meant to be simple byte-strings without endianess (see below for their ordering in practice).

Endianess of Hardware Channels
The endiness of the Hardware Channel numbers in Channel Map is same as in Port 2188h/218Eh. The endianess of the fixed values (0121h and 0124h) is also same as Port 2188h/218Eh. Ie. one can use that values without needing to swap LSBs and MSBs.

Endianess of Software Channels
The fixed 4-byte pairs are using same byte-order as how they are ordered in Channel Map (for example, 1.2.0.48 means that 48 (30h) is at highest address). So far it's simple. The (slightly) confusing part is that SNES software usually encodes them as 2-word pairs (since the SNES uses a little-endian CPU, the above example values would be 0201h.3000h).

SNES Cart Satellaview Town Status Packet

Town Status (Software Channel 1.1.0.5)
Loaded to 7EA31Dh, then copied to 7EA21Dh. Size (max) 256 bytes.
Uses a normal 10-byte fragment header, but with "4bit Transmission ID" and "7bit Fragment Number" ignored, but still does use "Target Offset" for fragment(s)?
```
  00h   1   Flags (bit0=1=Invalid) (bit1-7=unknown/unused)
  01h   1   Town Status ID (packet must be processed only if this ID changes)
  02h   1   Directory ID   (compared to Directory ID in Directory packet)
  03h   4   Unknown/unused
  07h   1   APU Sound Effects/Music & BSX Receiver Power-Down
             Bit0-3 Unknown/unused
             Bit4-5 APU (0=Mute, 1=Effects, 2=Effects/MusicA, 3=Effects/MusicB)
             Bit6   BSX (0=Normal, 1=Power-down with Port 2199h Reg[0]=88h)
```

```
            Bit7   BSX (0=Normal, 1=Power-down with Port 2199h Reg[0]=00h)
            (Or, maybe, the "Power-down" stuff enables satellite radio,
            being injected to audio-inputs on expansion port...?)
  08h   1   Unknown/unused
  09h   8   People Present Flags (Bit0-63) (max 5)        (LITTLE-ENDIAN)
  11h   2   Fountain Replacement & Season Flags (Bit0-15) (LITTLE-ENDIAN)
  13h   4   Unknown/unused
  17h   1   Number of File IDs (X) (may be 00h=none) (max=E8h)
  18h   X   File IDs (one byte each) (compared against File ID in Directory)
```

This packet should be (re-)downloaded frequently. The File IDs indicate which Directory entries are valid (whereas, it seems to be possible to share the same ID for ALL files), the Directory ID indicates if the Directory itself is still valid.

Fountain/Replacement and Season

The animated Fountain (near beach stairs) has only decorative purposes (unlike buildings/people that can contain folders). Optionally, the fountain can be replaced by other (non-animated) decorative elements via Fountain Replacement Flags in the Town Status packet: Bit0-11 are selecting element 1-12 (if more than one bit is set, then the lowest bit is used) (if no bits are set, then the fountain is used).

```
  None  00h Default Fountain (default when no bits set) (animated)
  Bit0  01h Jan     Altar with Apple or so
  Bit1  02h Feb     Red Roses arranged as a Heart
  Bit2  03h Mar     Mohican-Samurai & Batman-Joker
  Bit3  04h Apr     Pink Tree
  Bit4  05h May     Origami with Fish-flag
  Bit5  06h Jun     Decorative Mushrooms
  Bit6  07h Jul     Palmtree Christmas with Plastic-Blowjob-Ghost?
  Bit7  08h Aug     Melons & Sunshade
  Bit8  09h Sep     White Rabbit with Joss Sticks & Billiard Balls
  Bit9  0Ah Oct     National Boule-Basketball
  Bit10 0Bh Nov     Red Hemp Leaf (cannabis/autum)
  Bit11 0Ch Dec     Christmas Tree (with special Gimmick upon accessing it)
```

As shown above, the 12 replacements are eventually associated with months (Christmas in December makes sense... and Fish in May dunno why).
Bit12-15 of the above flags are selecting Season:

```
  None  00h Default (when no bits set)
  Bit12 01h Spring  (pale green grass) (seems to be same as default)
  Bit13 02h Summery (poppy colors with high contrast)
  Bit14 03h Autumn  (yellow/brown grass)
  Bit15 04h Winter  (snow covered)
```

As for the Fountain, default is used when no bits set, and lowest bit is used if more than one bit is set.

SNES Cart Satellaview Directory Packet

Directory (Software Channel 1.1.0.6)

Loaded to 7FC000h and then copied to 7EC000h. Size (max) 16Kbytes.

```
  00h   1   Directory ID (compared to Directory ID in Town Status packet)
  01h   1   Number of Folders (Buildings, People, or hidden folders)
  02h   3   Unknown/unused
  05h   ..  Folders (and File) entries (see below)       ;\together
  ..    1   Number of Expansion Data Entries (00h=none)  ; max 3FFBh bytes
  ..    ..  Expansion Data Entries (see next chapter)    ;/
```

Folder Entry Format

```
  00h   1   Flags (bit0=1=Invalid) (bit1-7=unknown/unused)
  01h   1   Number of File Entries (if zero: buildings are usually closed)
  02h   15h Folder Name (max 20 chars, terminated by 00h) (shown in building)
  17h   1   Length of Folder Message (X) (in bytes, including ending 00h)
  18h   X   Folder Message/Description (terminated by 00h)
  18h+X 1   More Flags (Folder Type)
             Bit0   : Folder Content (0=Download/Files, 1=Shop/Items)
             Bit1-3 : Folder Purpose (0=Building, 1=Person, 2=Include-Files)
               000b = Indoors (Building ID at [19h+X])     (Bit3-1=000b)   (0)
               x01b = Outdoors (Person ID at [19h+X])      (Bit2-1=01b)  (1,5)
               x1xb = Folder contains hidden Include Files (Bit2=1b) (2,3,6,7)
               100b = Unknown/unused (may be useable for "Files at Home"?) (4)
             Bit4-7 : Unknown/unused
  19h+X 1   Folder 6bit ID (eg. for Building:01h=News, for People:01h=Hiroshi)
  1Ah+X 1   Unknown/Unused
  1Bh+X 1   Unknown/Unused
  1Ch+X 1   Clerk/Avatar (00h..10h) (eg. 0Eh=Robot, 10h=BS-X) (11h..FFh=Crash)
  1Dh+X 1   Unknown/Unused
  1Eh+X 1   Unknown/Unused
  1Fh+X 1   Unknown/Unused
  20h+X ..  File/Item Entries (each one is 32h+X bytes; for items: fixed X=79h)
```

File/Item Entry Format

For Both Files and Items:

```
  00h   1   File ID (compared to File IDs in Town Status Packet)
  01h   1   Flag (bit0=used by "Town Status" check (1=Not Available))
  02h   15h File Name (max 20 chars, terminated by 00h) (shown in building)
```

For Files: (in File Folders)

```
  17h   1   Length of File Message (X) (in bytes, including ending 00h)
  18h   X   File Message/Description (terminated by 00h)
```

For Items: (in Item Folders)

```
  17h   1   Length of Item Description+Activation+Price+Flag (X) (fixed=79h)
  18h   25h Item Description (max 36 chars, plus ending 00h)
  3Dh   47h Item Activation Message (min 1, max 70 chars, plus ending 00h)
  84h   12  Item Price (12-Digit ASCII String, eg. "000000001200" for 1200G)
  90h   1   Item Drop/Keep Flag (00h=Drop after Activation, 01h=Keep Item)
```

For Both Files and Items:

```
  18h+X 4   Software Channel Number of Current File (for Items: N.N.0.0=None)
  1Ch+X 3   Big-Endian Filesize
  1Fh+X 3   Unknown/unused (except, first 2 bytes used by Derby Stallion 96)
  22h+X 1   Flags
              Bit0: used by "Town Status" check (1=Not Available)
              Bit1: Unknown/unused
              Bit2: Building Only (0=Also available at Home, 1=Building only)
              Bit3: Low Download Accuracy / Streaming or so
               0=High-Download-Accuracy (for programs or other important data)
```

```
             1=Low-Download-Accuracy (for audio/video data streaming)
            Bit4: Unused (except, must be 0 for Derby Stallion 96 files)
            Bit5-7: Unknown/unused
  23h+X 1   Unknown/unused
  24h+X 1   Flags/Target (seems to be same/similar as in Channel Map)
             Bit2-3:
              0=Download to WRAM (not really implemented, will crash badly)
              1=Download to PSRAM (without saving in FLASH)
              2=Download to Continous FLASH banks (erases entire chip!)
              3=Download to FREE-FLASH banks (relocate to PSRAM upon execution)
  25h+X 2   Unknown/unused
  27h+X 1   Date (Bit7-4=Month, Bit3-0=0)            ;\copied to Satellaview
  28h+X 1   Date (Bit7-3=Day, Bit2=0, Bit1-0=Unknown) ;/FLASH File Header FFD6h
  29h+X 1   Timeslot (Bit7-3=StartHours.Bit4-0, Bit2-0=Start Minutes.Bit5-3)
  2Ah+X 1   Timeslot (Bit4-0=EndHours.Bit4-0,   Bit7-5=Start Minutes.Bit2-0)
  2Bh+X 1   Timeslot (Bit7-2=EndMinutes.Bit5-0, Bit1-0=Unused)
  2Ch+X 4   Software Channel Number for additional Include File (N.N.0.0=None)
  30h+X 2   Unknown/unused
```

The directory may contain files that aren't currently transmitted; check the Town Status for list of currently available File IDs. Also check the Directory ID in the Town Status, if it doesn't match up with the ID of the Directory packet, then one must download the Directory again (otherwise one needs to download it only once after power-up).

Include Files

Each File in the directory has an Include File entry. Before downloading a file, the BIOS checks if the Include File's Software Channel Number is listed in the Directory (in any folder(s) that are marked as containing Include Files). If it isn't listed (or if it's N.N.0.0), then there is no include file. If it is listed, then the BIOS does first download the include file, and thereafter download the original file. Whereas, the include file itself may also have an include file, and so on (the download order starts with the LAST include file, and ends with the original file).

There are some incomplete dumps of some games, which have 1Mbyte FLASH dumped, but do require additional data in WRAM/PSRAM:

```
  BS Dragon Quest (copy of channel_map in PSRAM, episode_number in WRAM)
  BS Zelda no Densetsu Kodai no Sekiban Dai 3 Hanashi (hw_channel in WRAM)
```

The missing data was probably transferred in form of Include files.

Streaming Files

There seems to be some streaming support:

Files flagged as FILE[22h+X].Bit3=1 are repeatedly downloaded-and-executed again and again (possibly intended to produce movies/slideshows) (details: max 256K are loaded to upper half of PSRAM, and, if successfully received: copied to lower 256K of PSRAM and executed in that place; whereas, the execution starts once when receiving the next streaming block, that is: with a different 4bit transmission ID in the 10-byte packet header).

Moreover, Packets marked as having 00h fragments, are treated as having infinite fragments (this would allow to overwrite received data by newer data; though none of the higher-level BIOS functions seems to be using that feature).

Files Available at Home

Some files can be downloaded at the "Home" building (via the 3rd of the 4 options in that building), these "Home" files may be located in any folders, namely, including following folders:

```
  FOLDER[00h].Bit0=Don't care (folder may be marked as hidden)
  FOLDER[18h+X]=Don't care    (folder may be also used as building/person/etc.)
```

For downloading them at "Home", the files must be defined as so:

```
  FILE[1Ah+X]<>0000h   (software channel isn't N.N.0.0)
  FILE[22h+X].Bit2=0   (flagged as available at home)
  FILE[18h]=Don't care (file description isn't shown at home)
```

BUG: If there are more than 32 of such "Home" files, then the BIOS tries to skip the additional files, but destroys the stack alongside that attempt.

Note: Unlike other downloads, Home ones are always having a wooden-plank background, and are done without transmission Interval timeouts. And, Include Files are ignored. And, Autostart is disabled (ie. downloads to PSRAM are totally useless, downloads to FLASH can/must be manually started).

SNES Cart Satellaview Expansion Data (at end of Directory Packets)

Directory (Software Channel 1.1.0.6)

Loaded to 7FC000h and then copied to 7EC000h. Size (max) 16Kbytes.

```
  00h   1   Directory ID (compared to Directory ID in Town Status packet)
  01h   1   Number of Folders (Buildings, People, or hidden folders)
  02h   3   Unknown/unused
  05h   ..  Folders (and File) entries (see previous chapter) ;\together
  ..    1   Number of Expansion Data Entries (00h=none)       ; max 3FFBh
  ..    ..  Expansion Data Entries                            ;/bytes
```

Expansion Data Entry Format (after folder/file area)

```
  00h   1   Flags (bit0=1=Invalid) (bit1-7=unknown/unused)
  01h   1   Unknown/unused
  02h   2   Length (N) (16bit) (this one is BIG-ENDIAN)
  04h   N   Expansion Chunk(s) (all values in chunks are LITTLE-ENDIAN)
```

For for some reason, there may be more than one of these entries, but the BIOS does use only the first entry with [00h].Bit0=0=Valid (any further entries are simply ignored).

Chunk 00h (End)

```
  00h       1   Chunk ID (00h)
```

Ends (any further following chunks are ignored). Chunk 00h should be probably always attached at the end of the chunk list (although it can be omitted in some cases, eg. after Chunk 02h).

Chunk 01h (Custom Building)

All values in this chunk are LITTLE-ENDIAN.

```
  00h       1   Chunk ID (01h)
  01h       2   Chunk Length (73h+L1..L5) (LITTLE-ENDIAN)
  03h       11h Message Box Headline (max 16 chars, terminated by 00h)      ;\
  14h       20h BG Palette 5 (copied to WRAM:7E20A0h/CGRAM:50h)  (16 words) ;
  34h       38h BG Data (copied to WRAM:7E4C8Ch/BG1 Map[06h,0Dh])(4x7 words);/
  6Ch       2   Length of BG Animation Data (L1)                           ;\
  6Eh       L1  BG Animation Data (for Custom Building) (see below)        ;/
  6Eh+L1    2   Tile Length (L2) (MUST be nonzero) (zero would be 64Kbytes) ;\
                BUG: Below Data may not start at WRAM addr 7Exx00h/7Exx01h  ;
                (if so, data is accidently read from address-100h)          ;
  70h+L1    L2  Tile Data (DMAed to VRAM Word Addr 4900h) (byte addr 9200h) ;/
  70h+L1+L2 2   Length (L3) (should be even, data is copied in 16bit units) ;\
  72h+L1+L2 L3  Cell to Tile Xlat (copied to 7E4080h, BUG:copies L3+2 bytes);/
```

```
  72h+L1..L3 2   Length (L4) (MUST be even, MUST be nonzero, max 30h)       ;\
  74h+L1..L3 L4  Cell Solid/Priority List (copied to 7E45D0h,copies L4 bytes);/
  74h+L1..L4 2   Unknown/unused, probably Length (L5)                       ;\
  76h+L1..L4 L5  Door Location(s) (byte-pairs: xloc,yloc, terminated FFh,FFh);/
                 Door Locations work only if BG1 cells also have bit15 set!
                 Bit15 must be set IN FRONT of the door, this is effectively
                 reducing the building size from 4x7 to 4x6 cells.
                 Note: Animated BG cells are FORCEFULLY having bit15 cleared!
```

This chunk may be followed by Chunk 02h. If so, it processes Chunk 02h (and ends thereafter). Otherwise it ends immediately (ie. when the following Chunk has ID=00h) (or also on any "garbage" ID other than 02h).

Chunk 02h (Custom Persons)

```
  00h        1   Chunk ID (02h)
  01h        2   Chunk Length (N) (LITTLE-ENDIAN) (N may be even,odd,zero)
  03h        N   Data (copied to 7F0000h) (N bytes) (max 0A00h bytes or so)
                  00h 4  7F0000h  Person 2Ch - Token Interpreter Entrypoint
                  04h 4  7F0004h  Person 2Dh - Token Interpreter Entrypoint
                  08h .. 7F0008h  General Purpose (Further Tokens and Data)
```

Copies the Data, and ends (any further following chunks are ignored). If Person 2Ch/2Dh are enabled in the Town Status packet, then the person thread(s) are created with above entrypoint(s). The threads may then install whatever OBJs (for examples, see the table at 99DAECh, which contains initial X/Y-coordinates and Entrypoints for Person 00h..3Fh).

Chunk 03h..FFh (Ignored/Reserved for future)

```
  00h        1   Chunk ID (03h..FFh)
  01h        2   Chunk Length (N) (LITTLE-ENDIAN)
  03h        N   Data (ignored/skipped)
```

Skips the data, and then goes on processing the following chunk.

BG Animation Data (within Chunk 01h) (for Custom Building)

```
  00h     2   Base.xloc (0..47) (FFFFh=No Animation Data)
  02h     2   Base.yloc (0..47)
  04h     X*4 Group(s) of 2 words (offset_to_frame_data,duration_in_60hz_units)
  04h+X*4 2   End (FFFEh=Loop/Repeat animation, FFFFh=Bugged/One-shot animat.)
  06h+X*4 ..  BG Animation Frame Data Block(s) (see below)
  ..      0-2 Padding (to avoid 100h-byte boundary BUG in L2)
```

Xloc/Yloc should be usually X=0006h,Y=000Dh (the location of the Custom Building, at 7E4C8Ch) (although one can mis-use other xloc/yloc values to animate completely different map locations; this works only if the Custom Building's folder contains files/items).
BUG: The one-shot animation is applied ONLY to map cells that are INITIALLY visible (ie. according to BG scroll offsets at time when entering the town).

BG Animation Frame Data Block(s)

```
  00h     Y*8 Group(s) of 4 words (xloc, yloc, bg1_cell, bg2_cell)
  00h+Y*8 2   End of Frame List (8000h) (ie. xloc=8000h=end)
```

NOTE: offset_to_frame_data is based at [94] (whereas, [94] points to the location after Chunk 01h ID/Length, ie. to the base address of Chunk 01h plus 3).
If the animation goes forwards/backwards, one may use the same offsets for both passes.
The Custom Bulding has 4x7 cells in non-animated form (so, usually one should use xloc=0..3, yloc=0..6). Cells can be FFFFh=Don't change (usually one would change only BG1 foreground cells, and set BG2 background cells to FFFFh). For animated BG1 cells, Bit10-15 are stripped for some stupid reason (in result, animated BG1 calls cannot be flagged as Doors via Bit15).

BG Cell to Tile Translation

Each 16x16 pixel Cell consists of four 8x8 Tiles. The Translation table contains tiles arranged as Upper-Left, Lower-Left, Upper-Right, Lower-Right. The 16bit table entries are 10bit BG tile numbers, plus BG attributes (eg. xflip).

Building/Map Notes

The PPU runs in BG Mode 1 (BG1/BG2=16-color, BG3=4-color). VRAM used as so:

```
  BG1 64x32 map at VRAM:0000h-07FFh, 8x8 tiles at VRAM:1000h-4FFFh (foreground)
  BG2 64x32 map at VRAM:0800h-0FFFh, 8x8 tiles at VRAM:1000h-4FFFh (background)
  BG3 32x32 map at VRAM:5000h-53FFh, 8x8 tiles at VRAM:5000h-6FFFh (menu text)
  OBJ 8x8 and 16x16 tiles at VRAM:6000h-7FFFh (without gap)        (people)
  Custom BG1/BG2 Tiles are at VRAM:4900h-xxxxh (BG.Tile No 390h-xxxh)
  Custom OBJ Tiles at VRAM:7C00h-xxxxh (OBJ.Tile No 1C0h..xxxh)
```

The PPU BG Palettes are:

```
  BG.PAL0 Four 4-Color Palettes (for Y-Button BG3 Menu)
  BG.PAL1 Four 4-Color Palettes (for Y-Button BG3 Menu)
  BG.PAL2 Buildings
  BG.PAL3 Buildings
  BG.PAL4 Buildings
  BG.PAL5 Custom Palette
  BG.PAL6 Landscape (Trees, Phone Booth)         (colors changing per season)
  BG.PAL7 Landscape (Lawn, Streets, Water, Sky)  (colors changing per season)
```

The town map is 48x48 cells of 16x16 pix each (whole map=768x768pix).

```
  Custom BG1/BG2 Cells are at WRAM:7E4080h-7E41FFh (Cell No 3D0h-3FFh)
  Custom Cell Solid/Priority...
```

SNES Cart Satellaview Other Packets

SNES Patch Packet (by 105BBC) (Software Channel 1.1.0.7)

Loaded to 7FC000h, data portions then copied to specified addresses. Size (max) 16Kbytes.

```
  00h  1   Number of entries (01h..FFh) (MUST be min 01h)
  01h  ..  Entries (max 3FFFh bytes)
```

Each entry is:

```
  00h  1   Flags (bit0=1=Invalid) (bit1-7=unknown/unused)
  01h  1   Unknown/unused
  02h  2   Length (N) (16bit, big-endian) (MUST be min 0001h)
  04h  3   SNES-specific Memory Address (24bit, big-endian)
  07h  N   Data (N bytes)
```

The data portions are copied directly to the specified SNES addresses (the BIOS doesn't add any base-offset to the addresses; ie. if the data is to be copied to WRAM, then the satellite would transmit 7E0000h..7FFFFFh as address; there is no address checking, ie. the packet can overwrite stack or I/O ports).

Welcome Message Packet (Software Channel 1.1.0.4)

Loaded to 7EA1B9. Size max 64h bytes (100 decimal). Displayed in text window with 37x4 ASCII cells.

```
  00h  100 Custom Message (max 99 characters, plus ending 00h)
```

If this packet is included in the Channel Map (and if it's successfully received), then the Custom Message is displayed right before entering the town. The japanese

Default Message is still displayed, too (so one gets two messages - which is causing some annoying additional slowdown).

Time Channel Packet (Software channel 1.1.0.8) (BS Fire Emblem)
Time Channel Packet (Software channel 1.1.0.8) (BS Satella2 1)
Time Channel Packet (Software channel 1.1.0.8) (BS Parlor Parlor 2)
Time Channel Packet (Software channel 1.1.0.8) (BS Shin Onigashima 1)
Time Channel Packet (Software channel 1.1.0.8) (BS Tantei Club)
Time Channel Packet (Software channel 1.1.0.8) (BS Kodomo Tyosadan..)
Time Channel Packet (Software channel 1.1.0.8) (BS Super Mario USA 3)
Time Channel Packet (Software channel 1.1.0.8) (BS Super Mario Collection 3)
Time Channel Packet (Software channel 1.1.0.8) (BS Excitebike - Mario .. 4)
Time Channel Packet (Software channel 1.1.0.8) (Itoi Shigesato no Bass Tsuri)
Time Channel Packet (Software channel 1.2.0.48) (BS Dragon Quest 1)
Time Channel Packet (Software channel 1.2.0.48) (BS Zelda .. Remix)
Time Channel Packet (HW channel [7FFFF7h]) (BS Zelda - Kodai .. Dai 3 ..)
Time Channel Packet (HW channel [7FFFF7h]) (BS Marvelous Camp Arnold 1)
Time Channel Packet (HW channel [7FFFF7h]) (BS Marvelous Time Athletic 4)
Preceeded by a 10-byte packet header. Of which, first 5 bytes are ignored (Body is 8-bytes, so packet size should be probably 5+8). Fixed 01h must be 01h. Number of Fragments must be 01h. Target Offset must be 000000h. The 8-byte Data Body is then:

```
00h  1  Unknown/unused (probably NOT seconds) ;(un-)used by Itoi only
01h  1  Minutes     (0..3Bh for 0..59)
02h  1  Hours       (0..17h for 0..23) (or maybe 24..26 after midnight)
03h  1  Day of Week (01h..07h) (rather not 00h..06h) (?=Monday)
04h  1  Day         (01h..1Fh for 1..31)
05h  1  Month       (01h..0Ch for 1..12)
06h  1  Unknown/unused (maybe year)    ;\could be 2x8bit (00:00 to 99:99)
07h  1  Unknown/unused (maybe century) ;/or maybe 16bit (0..65535) or so
```

Caution: The BS satellite program specified hours from 11..26 (ie. it didn't wrap from 23 to 0), the Time Channel(s) might have been following that notation; if so, then Date values might have also been stuck on the previous day. Unknown if the Time Channels have been online 24 hours a day, or if their broadcast ended at some time late-night.
Time Channel are often used by Soundlink games, in so far, it's also quite possible that the "hours:minutes" are referring to the time within the broadcast (eg. from 0:00 to 0:59 for a 1-hour broadcast duration), rather than to a real-time-clock-style time-of-the-day.
Differences between 1.1.0.8 and 1.2.0.48 are unknown. Some games require incoming Hardware channel number at [7FFFF7h] (from a separate loader or so).
One would expect TIME[0] to contain seconds - however, there aren't any games using that entry as seconds (instead, they wait for minutes to change, and reset seconds to zero; due to that wait-for-next-minute mechanism, many BSX games seem to "hang" for up to 60 seconds after booting them).
TIME[4..7] aren't really used as "date" by any games (a few games seem to be using TIME[4] or TIME[5] to determine how often the user has joined the game). Itoi uses (and displays) TIME[4..5] as date.
Some games are treating TIME[6..7] as a 16bit little-endian value, others are using only either TIME[6] or TIME[7] (ie. half of the games that use the "year" values are apparently bugged).

File Packet (Software Channel N.N.N.N as taken from Directory)

```
00h  N   Data, N bytes (N=Filesize as specified in Directory)
```

The file must contain a Satellaview Transmit Header (at file offset 7Fxxh or FFxxh), that header is similar to the FLASH File Header. For details & differences, see:
SNES Cart Satellaview FLASH File Header
The filesize should be usually a multiple of 128Kbytes (because the checksum in File Header is computed accross that size). Transmitting smaller files is possible with some trickery: For FLASH download, assume FLASH to be erased (ie. expand checksum to FFh-filled 128Kbyte boundary). For PSRAM download, the checksum isn't verified (so no problem there). Moreover, filesize should be usually at least 32Kbytes (for LoROM header at 7Fxxh), however, one could manipulate the fragment-offset in packet header (eg. allowing a 4Kbyte file to be loaded to offset 7000h..7FFFh).

Special Channel(s) used by Itoi Shigesato no Bass Tsuri No. 1 (1.2.130.N)
Itoi supports four special channel numbers to unlock special contests. The game doesn't try to receive any data on that channels (it only checks if one of them is listen in the Channel Map).

```
1.2.130.0      ;\Special Contests 1..4 (or so)
1.2.130.16     ; The 4 contests are looking more or less the same, possibly
1.2.130.32     ; with different parameters, different japanese descriptions,
1.2.130.48     ;/contest 2-3 have "No fishing" regions in some lake-areas.
1.2.130.other  ;-Invalid (don't use; shows error with TV-style "test screen")
```

For Itoi, the Channel Map may be max 1019 bytes (or even LESS?!) (unlike usually, where it could be 1485 bytes). There should be only one channel with value 1.2.130.N in the Channel Map. The game additionally requires the 1.1.0.8 Time Channel. And, uses the "APU" byte in the Town Status packet.

1.2.129.0 Special Channel used by Derby Stallion 96 (Dish on Building/Roof)
1.2.129.16 Special Channel used by Derby Stallion 96 (6th Main Menu Option)
Differences between the two channels are unknown; both are processed by the same callback function (and thus seem to have the same data-format). The packet(s) are both loaded to 7F0000h, size could be max 8000h (although, actual size might be 7E00h, as indicated by the checksum calculation). The overall format is:

```
0000h 3    Unknown/unused (3 bytes)
0003h 8    ID "SHVCZDBJ" (compared against [B289D6])
000Bh 2    Number of Chunks at 0010h and up (16bit) (little-endian) (min 1)
000Dh 1    Unknown/unused? (8bit)
000Eh 1    Checksum (bytes at [0000h..7DFFh] added together) (8bit)
000Fh 1    Checksum complement (8bit)
0010h DF0h Chunks
7E00h 200h Begin of non-checksummed area (IF ANY) (if it DOES exist,
           then it MIGHT contain a file-style header at 7FB0h..7FFFh ?)
```

Note: The Chunks are processed via function at B28A10h. Despite of the hardcoded channel numbers, the packets must be also listed (with the same channel numbers) in the Directory Packet (as hidden Include File entries or so).

Hardware Channel 0121h - Test Channel
Used for hardware-connection test (received data is ignored). Used by BIOS function 105B6Ch; which is used only when pressing X+L+R buttons while the Welcome Message is displayed. Transmission Timeout for the Test Channel is 10 seconds.

SNES Cart Satellaview Buildings

Home Building (Starting Point)
This building can be entered at any time, the four japanese options are:

```
1) Load File from FLASH Card
2) Delete File from FLASH Card
```

```
3) Download File (only files that are "Available at Home") (max 32 files)
4) Delete Settings in SRAM
```

Buildings

The buildings are numbered roughly anti-clockwise from 00h..1Fh, starting at lower-left of the town map:

```
00h Robot Skyscraper (lower-left)
01h News Center
02h Parabol Antenna
03h Junkfood
04h Police
05h Maths +-x/
06h Beach Shop (Shop for Predefined ROM Items)
07h Turtle-Arena
08h C-Skyscaper (Shop for Predefined ROM Items)
09h Red-Heart Church
0Ah Red \\\ Factory (upper-right corner)
0Bh Dracula Gift-Shop
0Ch Cow-Skull Church
0Dh Spintop/Abacus (near Maths +-X/)
0Eh Blank Skyscraper (near Parabol Antenna)
0Fh Sign (near Red Factory) (works only ONCE)     (or custom building)
10h Greek Buddah Temple (upper-end)
11h Bigger Neighbor's Building
12h Smaller Neighbor's Building (unknown how to get in there)
13h Phone Booth (can be entered only with Telephone Card item)
14h Sewerage (near Spintop) (Shop for Predefined ROM Items)
15h Unused
16h Unused ;\these Building-Folders MUST not exist (else BIOS randomly
17h Unused ;/crashes, accidently trying to animate Building number "44h/3")
18h Special Location without folder: Player's Home
19h Special Location without folder: Hydrant (near police)
1Ah Special Location without folder: Talking Tree (near C-Skyscraper)
1Bh Special Location without folder: Fountain (or Fountain Replacement)
1Ch Special Location without folder: Beach Toilets (Railway Station)
1Dh Special Location without folder: Ocean's Shore
1Eh Special Location without folder: Unused
1Fh Special Location without folder: Unused
20h-3Fh Building-Folders with these IDs do destroy memory (don't use!)
```

Buildings can be entered only if there is a corresponding folder (in the directory), and only if the folder contains at least one file or item (for the three pre-defined Shops it works also if the folder is empty).

SNES Cart Satellaview People

People

People are showing up only if they are flagged in the 64bit People Present Flags in the Town Status packet. If more than 5 people are flagged, then only the first 5 are shown (regardless of that limit, the 4 frogs and the ship can additionally be there).

```
00h Red Ball (on beach) (disappears after access)
01h Spring-boot (aka Dr.Hiroshi's Shop) (near news center) (sells items)
02h General Pee (showing up here and there pissing against buildings)
03h Brown Barbarian on Cocaine (near temple)
04h Blue Depressive Barbarian (near temple)
05h Ghost Waver (near phone booth)
06h Boy in Neighborhood
07h Older Elvis (near churches)
08h Purple Helmet (on beach)
09h Surfer (near beach shop)
0Ah Grayhaired (northwest lawn)
0Bh Alien Man (near phone booth)
0Ch Uncle Bicycle (near parabol antenna)
0Dh Circus Man (near temple/lake)
0Eh Speedy Blind Man (near parabol antenna/spintop)
0Fh Blonde Boy (near factory)
10h Girl with Pink Dress (near Bigger Neighbor's Home)
11h Brunetty Guy (near Bigger Neighbor's Home)
12h Brunette (near junkfood)
13h Darkhaired (near junkfood)
14h Blue Longhair (near junkfood)
15h Brunette Longhair (near junkfood)
16h Brunette Longhair (near red-heart church)
17h Green Longhair (near red-heart church)
18h Bicycle Girl (near C-Skyscraper)
19h Brunette Office Woman (near C-Skyscraper)
1Ah Blue Longhair (near parabol antenna)
1Bh Turquoise Longhair (near home)
1Ch Blue Longhair (near maths/spintop)
1Dh Brunette Longhair (near news center/beach stairs)
1Eh Black Longhair (near police)
1Fh Red Longhair (southeast beach)
20h Blackhaired Girl (near police)
21h Greenhaired Girl (on bench between temple and lake)
22h Graybluehaired older Woman (east of C-skyscraper)
23h Darkhaired Housewife (near home)
24h Traditional Woman (west of Robot-Skyscraper)
25h Greenhaired Girl (near Turtle-Arena)
26h Pinkhaired Girl (near Cow-Skull Church)
27h Brown Dog (northeast lawn)
28h White Dog (near home)
29h Gray Duck (near temple/lake)
2Ah Portable TV-Headed Guy (near Robot-Skyscraper)
2Bh Satellite Wide-screen TV-Headed Guy (near Robot-Skyscraper)
2Ch Custom Person 2Ch  ;\may be enabled only if defined in Expansion Area
2Dh Custom Person 2Dh  ;/of Directory Packet (otherwise crashes)
```

Below 2Eh-37h are specials which cannot have a Folder assigned to them:

```
2Eh Dead Dentist (on bench near lake) (gives Money when owning Fishing Pole)
2Fh Gimmick: Allows to use Bus/Taxi/Ferrari Tickets at Fountain
30h Gimmick: Allows to use Express/Museum-Train-Tickets at Railways Station
31h Gimmick: Special Event when accessing the Hydrant
32h Frog 32h (west of Robot-Skyscraper)       ;Change Identity Item
```

```
  33h Frog 33h (west of Robot-Skyscraper, too)  ;Change GUI Border Scheme
              (or on street near turtle arena?)
  34h Frog 34h (northwest lawn)                 ;Change GUI Color Scheme
  35h Frog 35h (near Cow-Skull Church)          ;Change GUI Cursor Shape
  36h Gimmick: Allows to use Whale/Dolphin/Fish Food at Oceans Shore
  37h Ship (cannot be accessed?) (near factory)
  38h Mr.Money (near police) (donates a 500G coin)  ;\only one can be present
  39h Mr.Money (near police) (donates a 1000G coin) ; at once, after the coin,
  3Ah Mr.Money (near police) (donates a 5000G coin) ;/all do act as Folder 38h
  3Bh-3Fh Unused?
```

Like Buildings, People can have a Folder associated to them (using above values as Folder ID, at least, that works for People 00h..2Bh).
If there is no corresponding folder transmitted, then People aren't doing anything useful (aside from producing whatever japanese sentences). One exception: If there's no People-File-Folder with ID=01h, then the Spring-boot guy acts as "Dr Hiroshi's Shop" where one can buy question-marked items for 3000G. The Frogs can be picked up (adding an Item to the inventory). Frog positions seem to be random (west of Robot-Skyscraper, street near Turtle Arena, northwest lawn, or near Cow-Skull Church).

Avatars

These are assigned for folders (either for people in buildings, or people on the streets).

```
  00h Geisha (faecher)
  01h Snorty (nose bubble)
  02h Gold hat
  03h Naked guy
  04h Soldier (lanze)
  05h Whore (lipstick/eye blinking)
  06h Wise man1 (huge white eyebrows)
  07h Wise man2 (huge stirn, huge ears)
  08h DJ Proppy (headphones, sunglasses, muscles)
  09h Casino manager (fat guy with red slip-knot)
  0Ah Student girl (manga, karierte bluse)
  0Bh School girl (satchel ranzen/smiley)
  0Ch Kinky gay (green hair, sunglasses, gold-ohrring)
  0Dh Yankee (glistening teeth/dauerwelle)
  0Eh Robot (blech-mann)
  0Fh Blonde chick (blonde, lipstick)
  10h BS-X logo (not a person, just the "BS-X" letters)
  11h-FFh Unknown/Unused/Crashes
  30h None (seems to be an un-intended effect, probably unstable, don't use)
```

SNES Cart Satellaview Items

Predefined Items (and their 24bit memory pointers)

Items sold in C-Skyscraper:

```
 00 88C229h Transfer Device (allows to teleport to any building) (unlimited)
 01 88C2B8h Telephone Card (5) (allows to enter phone booth) ;\
 02 88C347h Telephone Card (4) (allows to enter phone booth) ; decreases
 03 88C3D6h Telephone Card (3) (allows to enter phone booth) ; after usage
 04 88C465h Telephone Card (2) (allows to enter phone booth) ;
 05 88C4F4h Telephone Card (1) (allows to enter phone booth) ;/
 06 88C583h Fishing Pole (allows to get Money from Dead Dentist, Person 2Eh)
 07 88C612h Express Train Ticket                 ;\these are treated special
 08 88C6A1h Museum Train Ticket                  ;/by code at 88936Ch
 09 88C630h Bus Ticket (at Fountain)    ;\these all have same description
 0A 88C7BFh Taxi Ticket                 ;
 0B 88C84Eh Ferrari Blowjob Ticket      ;/
```

Items sold by Dr.Hiroshi (spring-boot guy near News Center)

```
 0C 88C8DDh Doping Item (walk/run faster when pushing B Button)
 0D 88C96Ch Unknown (disappears after usage)
```

Items sold in Beach Shop:

```
 0E 88C9FBh Whale Food (can be used at Oceans Shore)
 0F 88CA8Ah Dolphin Food (can be used at Oceans Shore)
 10 88CB19h Fish Food (can be used at Oceans Shore)
```

Items sold in Sewerage:

```
 11 88CBA8h Boy/Girl Gender Changer (can be used only once)
 12 88CC37h Transform Boy/Girl into Purple Helmet guy (Person 08h)(temporarily)
 13 88CCC6h Transform Boy/Girl into Brunette chick    (Person 1Dh)(temporarily)
 14 88CD55h Smaller Neighbor's Home Door Key (allows to enter that building)
```

Items obtained when picking-up Frogs:

```
 15 88CDE4h Change Identity (edit user name) (from Frog 32h) (works only once)
 16 88CE73h Change GUI Border Scheme         (from Frog 33h) (works only once)
 17 88CF02h Change GUI Color Scheme          (from Frog 34h) (works only once)
 18 88CF91h Change GUI Cursor Shape          (from Frog 35h) (works only once)
```

Item Format

As shown above, 25 items are defined in ROM at 99C229h-88D020h with 8Fh bytes per item. Custom Items (defined in Directory packet's "File" entries) can be stored at 10506Ah. The item format is:

```
  00h 15h Item Name (max 20 chars, plus ending 00h) (First 2 bytes 00h = Free)
  15h 1   Length of following (Description, Pointer, Whatever) (always 79h?)
  16h 25h Item Description (max 36 chars, plus ending 00h)
  3Bh 47h Item Activation Message (max 70 chars, plus ending 00h)
   If Activation Message = empty (single 00h byte), then Item Function follows:
   3Ch 3   Pointer to Interpreter Tokens (eg. 99974Dh for Transfer Device)
   3Fh 43h Unknown/Unused/Padding (should be zero)
   (there is no SRAM allocated for custom item functions,
   so this part may be used only for predefined ROM items)
  82h 12  Item Price (12-Digit ASCII String, eg. "000000001200" for 1200G)
  8Eh 1   Item Drop/Keep Flag (00h=Drop after Activation, 01h=Keep Item)
```

In case of Custom Items, above ITEM[00h..8Eh] is copied from FILE[02h..90h] (ie. a fragment of "File" Entries in the Directory Packet).
Entry [15h] seems to be always 79h, giving a total length of 8Fh per item.
The Item Message is used for items that cannot be activated (eg. "You can't use telephone card outside of the phone booth.",00h). If the message is empty (00h), then the next 24bit are a pointer to the item handler (eg. the Teleport function for the Transfer Device).
Note: Items can be listed, activated, and dropped via Y-Button. The teleport device can be also activated via X-button.

Shops

There are four pre-defined shops: Dr.Hiroshi's appears when Person 01h exists, WITHOUT folder assigned, or WITH an item-folder. The Beach Shop, C-Skyscraper and Sewerage Shops appear if they HAVE an folder assigned, the folder must be flagged as Item-Shop. In all cases, the folder may contain additional

items which are added to the Shop's predefined item list. Custom Shops can be created by assigning Item-Folders to other People/Buildings (in that case, the Folder MUST contain at least one item, otherwise the BIOS shows garbage). Shops may contain max 0Ah items (due to 7E865Eh array size).

SNES Cart Satellaview SRAM (Battery-backed)

The Satellaview BIOS cartridge contains 32Kbyte battery-backed SRAM, mapped to eight 4Kbyte chunks at 5000h..5FFFh in Bank 10h..17h.

SRAM Map

```
  0000h 2    ID "SG" (aka 53h,47h)
  0002h 2    Checksum Complement (same as below checksum XORed with FFFFh)
  0004h 2    Checksum (bytes 0..2FFFh added together) (assume [2..5]=0,0,FF,FF)
  0006h 20   User's Name (Shift-JIS)
  001Ch 2    User's Gender (0000h=Boy, 0001h=Girl)
  001Eh 6    Money (max 00E8D4A50FFFh; aka 999,999,999,999 decimal)
  0024h 2    Number of Items (0..10h) (or temporarily up to 11h)
  0026h 44h  Item Entries (4-bytes each: Type=00/01=ROM/RAM, and 24bit pointer)
  006Ah 8F0h Custom RAM Items (8Fh bytes each) (First 2 bytes 00h = free entry)
  095Ah 2    Remaining Time on Doping Item (decreases when entering buildings)
  095Ch 2    Number of Doping Items (walk/run faster when pushing B Button)
  095Eh 2    Remaining Calls on first Telephone Card Item minus 1
  0960h 2    Number of Telephone Card Items (unlocks Phone Booth)
  0962h 2    Number of Transfer Devices (enables Teleport via menu or X-Button)
  0964h 2    Number of Fishing Poles (allows to get Money from Dead Dentist)
  0966h 2    Number of Smaller Neighbor's Home Keys (0000h=Lock, other=Unlock)
  0968h 2    GUI Cursor Shape 16bit selector (0000h..0005h) (other=crash)
  096Ah 3    GUI Border Scheme 24bit pointer (def=9498D9h) (MUST be 94xxxxh)
  096Dh 3    GUI Color Scheme 24bit pointer (initially 94A431h)
  0970h 1    Player got 500 coin from Person 38h      ;\(00h=No, 01h=Yes,
  0971h 1    Player got 1000 coin from Person 39h     ; flags stay set until
  0972h 1    Player got 5000 coin from Person 3Ah     ; that Person leaves
  0973h 1    Player picked-up Red Ball aka Person 00h ;/the town)
  0974h 18h  BIOS Boot/NMI/IRQ Hook Vectors (retf's) (mapped to 105974h and up)
  098Ch 2B0h BIOS Function Hook Vectors (jmp far's) (mapped to 10598Ch and up)
  0C3Ch 64h  BIOS Reset Function (and some zero-filled bytes)
  0CA0h 100h BIOS Interpreter Token Handlers (16bit addresses in bank 81h)
  0DA0h 100h Garbage Filled (reserved for unused Tokens number 80h..FFh)
  0EA0h 2    Garbage Filled (for impossible 8bit token number 100h)
  0EA2h 215Eh Reserved (but, mis-used for game positions by some games)
  3000h 3000h Backup Copy of 0..2FFFh
  6000h 2000h General Purpose (used for game positions by various games)
```

Game Positions in SRAM

```
  0000h-0EA1h  BX-X BIOS (see above)
  1400h-14FFh  BS Super Mario USA 3 (256 bytes)
  1500h-15FFh  BS Super Mario Collection 3 (256 bytes)
  1500h-15FFh  BS Kodomo Tyosadan Mighty Pockets 3 (256 bytes)
  1600h-1626h  BS Satella Walker 2 (27h bytes)
  1600h-1626h  BS Satella2 1 (27h bytes)
  1700h-17FFh  BS Excitebike Bun Bun Mario Battle Stadium 4 (256 bytes)
  2000h-27FFh  BS Marvelous Camp Arnold Course 1 (2Kbytes)
  2800h-2F89h  BS Dragon Quest 1 (1.9Kbytes) (probably 1K, plus 1K backup copy)
  2006h-2FF5h  BS Zelda no Densetsu Remix (3.9Kbytes)
  2000h-2EFFh  BS Zelda no Densetsu Kodai no Sekiban Dai 3 Hanashi (3.75K)
  2000h-2FFFh  BS Super Famicom Wars (V1.2) (first 4K of 8Kbytes)
  3000h-5FFFh  Backup Copy of 0..2FFFh (not useable for other purposes)
  6000h-63FFh  BS Treasure Conflix (1Kbyte)
  6020h-62F9h  BS Sutte Hakkun 98 Winter Event Version (0.7Kbytes)
  6000h-7FFFh  BS Chrono Trigger - Jet Bike Special (8Kbytes)
  6800h-6FFFh  BS Super Famicom Wars (V1.2) (middle 2K of 8Kbytes)
  7500h-7529h  BS Cu-On-Pa (2Ah bytes)
  7800h-7FFFh  BS Super Famicom Wars (V1.2) (last 2K of 8Kbytes)
  7826h-7827h  BS Dr. Mario (only 2 bytes used?)
 (7C00h-7FFFh) BS Radical Dreamers (default, if free, at 7C00h) (1Kbyte)
```

There is no filesystem with filenames nor SRAM allocation. Most games are using hardcoded SRAM addresses, and do overwrite any other data that uses the same addresses.
One exception is BS Radical Dreamers: The game searches for a free (zerofilled) 1K block (defaults to using 7C00h-7FFFh), if there isn't any free block, then it prompts the user to select a 1K memory block (at 6000h-7FFFh) to be overwritten.
Some games are saving data in PSRAM (eg. Zelda no Densetsu: Kamigami no Triforce, in bank 70h) rather than SRAM - that kind of saving survives Reset, but gets lost on power-off.
There aren't any known BS games that save data in FLASH memory. Some BS games are using passwords instead of saving.

BIOS Hooks

These are 4-byte fields, usually containing a "JMP 80xxxxh" opcode (or a "RETF" opcode in a few cases), changing them to "JMP 1x5xxxh" allows to replace the normal BIOS functions by updated functions that are installed in SRAM. Unknown if any such BIOS updates do exist, and at which SRAM locations they are intended/allowed to be stored.

SRAM Speed

The Satellaview SRAM is mapped to a FAST memory area with 3.58MHz access time (unlike ALL other SNES RAM chips like internal WRAM or external SRAM in other cartridges).
The bad news is that this wonderful FAST memory isn't usable: It's located in Bank 10h-17h, so it isn't usable as CPU Stack or CPU Direct Page (both S and D registers can access Bank 00h only). And, the first 24Kbytes are used as reserved (and checksummed) area.

SNES Cart Satellaview FLASH File Header

Satellaview FLASH File Header

Located at offset 7Fxxh or FFxxh in file, mapped to FFxxh in bank 00h.

```
  FFB0h 2  Maker Code (2-letter ASCII)                 ;\garbage when
  FFB2h 4  Program Type (00000100h=Tokens, Other=65C816) ; [FFDBh]=01h
  FFB6h 10 Reserved (zero)                             ;/
  FFC0h 16 Title (7bit ASCII, 8bit JIS (?), and 2x8bit SHIFT-JIS supported)
  FFD0h 4  Block Allocation Flags (for 32 blocks of 128Kbytes each) (1=used)
             Retail (demo) games usually have ffff here -- Uh ???
```

```
            (exception BS Camp Arnold Marvelous)       -- Uh ???
  FFD4h 2  Limited Starts (bit15=0=Infinite, otherwise bit14-0=Remaining Flags)
  FFD6h 1  Date (Bit7-4=Month, Bit3-0=0)             ;\copied to from
  FFD7h 1  Date (Bit7-3=Day, Bit2=0, Bit1-0=Unknown) ;/Directory Packet
  FFD8h 1  Map Mode (20h=LoROM, 21h=HiROM) (or sometimes 30h/31h)
  FFD9h 1  File/Execution Type
            Bit0-3  Unknown/unused (usually/always 0)
            Bit4    Receiver Power Down (0=No/Sound Link, 1=Power-Down 2197h)
            Bit5-6  Execution Area (0=FLASH, 1=Reloc FLASH-to-PSRAM, 2/3=Fail)
            Bit7    Skip the "J033-BS-TDM1 St.GIGA" Intro (0=Normal, 1=Skip)
  FFDAh 1  Fixed (33h)
  FFDBh 1  Unknown (usually 02h, sometimes 01h, or rarely 00h) (see FFBxh)
  FFDCh 2  Checksum complement (same as below, XORed with FFFFh)
  FFDEh 2  Checksum (all bytes added together; assume [FFB0-DF]=00h-filled)
  FFE0h 32 Exception Vectors (IRQ,NMI,Entrypoint,etc.) (for 65C816 code)
```

Entrypoint is at 800000h+[FFFCh] for 65C816 Machine Code, or at 400000h for Interpreter Tokens (ie. when [FFB2h]=00000100h, as used by few magazines like BS Goods Press 6 Gatsu Gou).
Caution: Machine Code programs are started WITHOUT even the most basic initialization (one of the more bizarre pieces: the download "screensaver" may leave HDMAs enabled when auto-starting a downloaded file).

Satellaview PSRAM File Header (download to PSRAM without saving in FLASH)
Basically same as FLASH headers. FFD9h.Bit7 (skip intro) seems to be ignored (accidently using FFD9h from FLASH cartridge instead from PSRAM?). The checksum isn't verified (FFDEh must match with FFDCh, but doesn't need to match the actual file content).

Satellaview Transmit Header (located at 7Fxxh or FFxxh in File)
Basically same as normal Satellaview FLASH File Header. However, after downloading a file (and AFTER storing it in FLASH memory), the BIOS does overwrite some Header entries:

```
  FFD0h  4-byte  Block Allocation field (set to whichever used FLASH Blocks)
  FFD6h  2-byte  Date field (set to Date from Satellite Directory Entry)
  FFDAh  1-byte  Fixed Value (set to 33h)
```

Since FLASH cannot change bits from 0 to 1 (without erasing), the above values must be FFh-filled in the transmitted file (of course, for the fixed value, 33h or FFh would work) (and of course, the FFh-requirement applies only to FLASH downloads, not PSRAM downloads).

Notes
The title can be 16 bytes (padded with 20h when shorter), either in 7bit ASCII, 8bit JIS (or so?), or 2x8bit SHIFT-JIS, or a mixup thereof. A few files are somewhat corrupted: Title longer than 16 bytes (and thereby overlapping the Block Allocation flags).
Limited Starts consists of 15 flags (bit14-0), if the limit is enabled (bit15), then one flag is changed from 1-to-0 each time when starting the file (the 1-to-0 change can be done without erasing the FLASH sector).
Note that the checksum excludes bytes at FFB0h-FFDFh, this is different as in normal cartridges (and makes it relative easy to detect if a file contains a SNES ROM-image or a Satellaview FLASH-file/image.
Files are treated as "deleted" if their Fixed Value isn't 33h, if Limited Starts is 8000h (limit enabled, and all other bits cleared), or if Checksum Complement entry isn't equal to Checksum entry XOR FFFFh. Some FLASH dumps in the internet do have experired Limited Starts entries (so they can be used only when changing [FFD4h] to a value other than 8000h).
The FLASH card PCBs can be fitted with 1/2/4 MByte chips (8/16/32 Mbit). As far as known, all existing FLASH cards contain 1MByte chips (so Block Allocation bit8-31 should be usually always 0). Most files are occupying the whole 1MByte (so bit0-7 should be usually all set). There are also some 256Kbyte and 512Kbyte files (where only 2 or 4 bits would be set). Minimum file size would be 128Kbyte. Odd sizes like 768Kbytes would be also possible.
Unlike the Satellite Packet Headers, the FLASH/Transmit-File header contains "normal" little-endian numbers.

SNES Cart Satellaview BIOS Function Summary

BIOS functions must be called with BIOS ROM enabled in Bank 80h-9Fh (in some cases it may be required also in Bank 00h-1Fh), and with DB=80h. Incoming/outgoing Parameters are passed in whatever CPU registers and/or whatever WRAM locations. WRAM is somewhat reserved for the BIOS (if a FLASH file changes WRAM, then it should preserve a backup-copy in PSRAM, and restore WRAM before calling BIOS functions) (WRAM locations that MAY be destroyed are 7E00C0h..7E00FFh and 7E1500h..7E15FFh, and, to some level: 7F0000h..7FFFFFh, which is used only as temporary storage).

Hooks (usually containing RETF opcodes)

```
  105974 boot_hook (changed by nocash fast-boot patch)
  105978 nmi_hook
  10597C irq_vector
  105980 download_start_hook --> see 9B8000
  105984 file_start_hook --> see 958000
  105988 whatever_hook --> see 99xxxx
```

SRAM Vectors

```
  10598C detect_receiver
  105990 port_2194_clr_bit0
  105994 port_2196_test_bit1
```

Copy Data Queue to RAM Buffer

```
  105998 set_port_218B_and_218C_to_01h
  10599C set_port_218C_to_00h
  1059A0 read_data_queue
```

Port 2199h (serial port 2) (maybe satellite audio related)

```
  1059A4 init_port_2199_registers
  1059A8 send_array_to_port_2199  ;BUGGED?
  1059AC recv_3x8bit_from_port_2199
  1059B0 send_16bit_to_port_2199
  1059B4 recv_8bit_from_port_2199
```

Port 2198h (serial port 1) (unused/expansion or so)

```
  1059B8 port_2198_send_cmd_recv_multiple_words
  1059BC port_2198_send_cmd_recv_single_word
  1059C0 port_2198_send_cmd_send_verify_multiple_words
  1059C4 port_2198_send_cmd_send_verify_single_word
  1059C8 port_2198_send_cmd_send_single_word
  1059CC port_2198_send_10h_send_verify_single_word
  1059D0 port_2198_send_cmd_verify_FFFFh
  1059D4 port_2198_send_20h_verify_FFFFh
  1059D8 recv_2198_skip_x BUGGED!
  1059DC recv_2198_want_x
  1059E0 send_30h_to_port_2198
  1059E4 send_00h_to_port_2198
  1059E8 send_8bit_to_port_2198
  1059EC wait_port_2198_bit7
```

Forward Data Queue from RAM to Target

```
1059F0 forward_data_queue_to_target
1059F4 forward_queue_to_wram
1059F8 forward_queue_to_psram
1059FC forward_queue_to_entire_flash
105A00 forward_queue_to_entire_flash_type1
105A04 forward_queue_to_entire_flash_type2
105A08 forward_queue_to_entire_flash_type3
105A0C forward_queue_to_entire_flash_type4
105A10 forward_queue_to_flash_sectors
105A14 forward_queue_to_flash_sectors_type1
105A18 forward_queue_to_flash_sectors_type2
105A1C forward_queue_to_flash_sectors_type3
105A20 forward_queue_to_flash_sectors_type4
105A24 forward_queue_to_channel_map  ;with 5-byte frame-header
105A28 forward_queue_to_town_status
```

FLASH Files

```
105A2C scan_flash_directory
105A30 allocate_flash_blocks
105A34 .. prepare exec / map file or so
105A38 verify_file_checksum
105A3C get_flash_file_header_a
105A40 delete_flash_file_a
105A44 get_flash_file_header_5A
105A48 copy_file_header
105A4C search_test_file_header, out:[57]
105A50 test_gamecode_field
105A54 copy_file_to_psram
105A58 get_file_size
105A5C decrease_limited_starts
```

Memory Mapping

```
105A60 map_flash_as_data_file  (for non-executable data-files?)
105A64 map_psram_as_data_file  (for non-executable data-files?)
105A68 .. mapping and copy 512Kbytes ?
105A6C map_flash_for_rw_access
105A70 map_flash_for_no_rw_access
105A74 map_flash_for_reloc_to_psram
105A78 .. mapping (unused?)
105A7C map_flash_as_lorom_or_hirom
105A80 execute_game_code
105A84 .. map_psram_for_streaming ???
105A88 map_psram_as_lorom_or_hirom
105A8C .. copy 256Kbytes...
```

FLASH Memory

```
105A90 flash_abort
105A94 flash_abort_type1
105A98 flash_abort_type2
105A9C flash_abort_type3
105AA0 flash_abort_type4
105AA4 flash_erase_entire
105AA8 flash_erase_entire_type1
105AAC flash_erase_entire_type2
105AB0 flash_erase_entire_type4 ;4!
105AB4 flash_erase_entire_type3
105AB8 flash_test_status   ERASE-PROGRESS
105ABC flash_test_status_type1
105AC0 flash_test_status_type2
105AC4 flash_test_status_type4 ;4!
105AC8 flash_test_status_type3
105ACC flash_erase_first_sector
105AD0 flash_erase_first_sector_type1
105AD4 flash_erase_first_sector_type2
105AD8 flash_erase_first_sector_type3
105ADC flash_erase_first_sector_type4
105AE0 flash_erase_next_sector
105AE4 flash_erase_next_sector_type1
105AE8 flash_erase_next_sector_type2
105AEC flash_erase_next_sector_type3
105AF0 flash_erase_next_sector_type4
105AF4 flash_write_byte
105AF8 flash_write_byte_type1
105AFC flash_write_byte_type2
105B00 flash_write_byte_type3
105B04 flash_write_byte_type4
105B08 flash_get_free_memory_size
105B0C flash_get_and_interprete_id
105B10 flash_get_id
105B14 flash_init_chip
105B18 flash_init_chip_type1
105B1C flash_init_chip_type2
105B20 flash_init_chip_type3
105B24 flash_init_chip_type4
```

Satellite Directory

```
105B28 apply_satellite_directory
105B2C directory_find_8bit_folder_id
105B30 directory_find_32bit_file_channel
105B34 test_if_file_available
105B38 download_file_and_include_files
105B3C directory_find_32bit_bugged
```

Misc...

```
105B40 .. initialize stuff on reset
105B44 download_nmi_handling (with download_callback etc.)
105B48 download_nmi_do_timeout_counting
105B4C nmi_do_led_blinking
105B50 mark_flash_busy
105B54 mark_flash_ready
105B58 set_port_2197_bit7
105B5C clr_port_2197_bit7
105B60 detect_receiver_and_port_2196_test_bit1
105B64 init_flash_chip_with_err_29h
105B68 init_flash_chip_with_err_2Ah
105B6C detect_receiver_and_do_downloads
```

```
  105B70 do_download_function
  105B74 retry_previous_download
  105B78 set_target_id_and_search_channel_map
  105B7C apply_target_for_download
  105B80 clear_queue_and_set_13D1_13D2
  105B84 flush_old_download   ;[218C]=0, clear some bytes
```
Invoke Download Main Functions
```
  105B88 download_to_whatever (BUGGED)
  105B8C download_channel_map
  105B90 download_welcome_message
  105B94 download_snes_patch
  105B98 download_town_status
  105B9C download_town_directory
  105BA0 download_to_memory
```
Download sub functions
```
  105BA4 add_download_array
  105BA8 wait_if_too_many_downloads
  105BAC do_download_callback
  105BB0 dload_channel_map_callback_1
  105BB4 dload_channel_map_callback_2
  105BB8 dload_welcome_message_callback
  105BBC dload_snes_patch_callback
  105BC0 dload_town_status_callback_1
  105BC4 dload_town_status_callback_2
  105BC8 dload_town_directory_callback_1
  105BCC dload_town_directory_callback_2
  105BD0 .. flash status
  105BD4 dload_to_mem_wram_callback1          ;\
  105BD8 dload_to_mem_wram_callback2          ;
  105BDC dload_to_mem_psram_callback1         ;
  105BE0 dload_to_mem_psram_callback2         ; dload_to_memory_callbacks
  105BE4 dload_to_mem_entire_flash_callback1  ;
  105BE8 dload_to_mem_entire_flash_callback2  ;
  105BEC dload_to_mem_free_flash_callback1    ;
  105BF0 dload_to_mem_free_flash_callback2    ;/
  105BF4 dload_to_mem_entire_flash_callback_final
  105BF8 dload_to_mem_free_flash_callback_final
  105BFC reset_interpreter_and_run_thread_958000h
  105C00 verify_channel_map_header
  105C04 raise_error_count_check_retry_limit
  105C08 search_channel_map
  105C0C post_download_error_handling
  105C10 .. erase satellite info ?
```
APU Functions
```
  105C14 apu_flush_and_clear_queues
  105C18 apu_flush_raw
  105C1C apu_message
  105C20 apu_nmi_handling
  105C24 apu_upload_extra_thread
  105C28 apu_upload_curr_thread
  105C2C apu_enable_effects_music_b
  105C30 apu_enable_effects_music_a
  105C34 apu_mute_effects_and_music
  105C38 apu_enable_effects_only
```
Reset
```
  105C3C reboot_bios (this one works even when BIOS=disabled or WRAM=destroyed)
```
Further Stuff
```
  105C96 Unused 7 bytes (used for nocash fast-boot patch)
  105C9D Unused 3 bytes (zero)
  105CA0 Token Vectors (16bit offsets in bank 81h)
```

BIOS Tables
```
  105xxx Tables in SRAM (see above)
  808000 Unsorted ptrs to BIOS Functions, Token-Extensions, and OBJ-Tile-Data
  9FFFF0 Pointers to source data for APU uploads
```

Additional BIOS Functions (without SRAM-Table vectors)

These are some hardcoded BIOS addresses (used by some FLASH programs).
```
  808C2A Invoke_dma_via_ax_ptr
  8091B6 Create_machine_code_thread
  809238 Pause_machine_code_thread
  80938F Do nothing (retf) (used as dummy callback address)
  80ABC8 ...whatever
  80AC01 ...whatever
  80B381 Upload_gui_border_shape_to_vram
  80B51B Clear_text_window_content
  80B91E Fill_400h_words_at_7E76000_by_0080h   ;clear whole BG3 map in WRAM
  80EB99 Injump_to_APU_Town_Status_handling (requires incoming pushed stuff)
  81C210 Reset_interpreter
  81C29A Set_interpreter_enable_flag
  81C2B0 Create_interpreter_thread
  81C80E Deallocate_all_obj_tiles_and_obj_palettes
```
Note: Some of the above functions are also listed in the table at 808000h.

Returning to BIOS

If an executable file wants to return control to BIOS, it must first reset the APU (if it has uploaded code to it), and then it can do one the following:
Perform a warmboot (the BIOS intro is skipped, but the Welcome message is re-displayed, and the player is moved back to the Home building):
```
  jmp 105C3Ch   ;srv_reboot_bios (simple, but quite annoying)
```
Or return to the BIOS NMI handler (from within which the executable was started) this is done by many games (player returns to the most recently entered building, this is more elegant from the user's view, though requires a messy hack from the programmer's view):
```
  call restore_wram            ;-restore WRAM (as how it was owned by BIOS)
  jmp  far (($+4) AND 00FFFFh)  ;-PB=00h (so below can map BIOS to bank 80h)
  mov  a,80h   ;\                                ;\
  push a ;=80h ;                                 ; set DB=80h, and
  pop  db      ;/                                ; enable BIOS in bank 80h-9Fh
  mov  [085000h],a ;map BIOS to bank 80h-9Fh     ; (though not yet in 00h-1Fh)
  mov  [0E5000h],a ;apply                        ;/
  call far 99D732h  ;super-slow ;out: M=0       ;-upload [9FFFF0h] to APU
 .assume p=10h  ;(above set M=0, and keeps X=unchanged)
```

```
  call far 81C210h                              ;-Reset Token Interpreter
  call far 81C29Ah                              ;-Enable/Unpause Interpreter
  call far 80937Fh ;set NMI callback to RETF prevent FILE to be executed AGAIN)
  mov  x,[13B2h]       ;BIOS online flag (8bit)  ;\skip below if offline
  jz   @@skip                                   ;/
  push pb       ;\retadr for below              ;\
  push @@back-1 ;/                              ;
  push db       ;-incoming pushed DB for below  ;
  push 7E00h    ;\                              ; init apu effects/music
  pop  db       ; incoming current DB for below ; (according to APU bits
  pop  db ;=7Eh ;/                              ; in town status packet)
  jmp  far 80EB99h ;--> injump to 105BC0h       ;
@@back:                                         ;
  .assume p=20h  ;(above set's it so)           ;
  ;(if executed, ie. not when @@skip'ed)        ;/
@@skip:
  clr  p,30h // .assume p=00h  ;below call 81C2B0h requires M=0, X=0
  mov  [0CDEh],0000h                            ;-mark fade-in/out non-busy
  mov  a,0099h     ;\                           ;\
  mov  [0BEh],a    ; 99D69A ;BIOS - enter town  ; create_interpreter_thread
  mov  a,0D69Ah    ;/                           ; (99D69Ah = enter town)
  call far 81C2B0h                              ;/
  set  p,20h // .assume p=20h
  mov  a,81h       ;\enable NMI and joypad (unstable: BIOS isn't yet mapped!)
  mov  [4200h],a   ;/caution: ensure that no NMI occurs in next few clk cycles
  mov  a,80h                                    ;\enable BIOS also in bank 0,
  mov  [075000h],a ;map BIOS to bank 00h-1Fh    ; and return to BIOS NMI handler
  jmp  far 80BC27h ;apply [0E5000h]=a, and retf ;/
```

SNES Cart Satellaview Interpreter Token Summary

Interpreter Tokens

```
  00h  ControlSubThread(pEntrypoint)  ;special actions upon xx0000h..xx0005h
  01h  SetXYsignViewDirectionToSignsOfIncomingValues(vX,vY) ;not if both zero
  02h  SleepWithFixedObjShape(wSleep,pObjShape)
  03h  SleepWithXYstepAs9wayObjShape(wSleep,pObjShape1,..,pObjShape9)
  04h  SleepWithXYsignAs9wayObjShape(wSleep,pObjShape1,..,pObjShape9)
  05h  ClearForcedBlankAndFadeIn(wSleep,wSpeedRange?)
  06h  MasterBrightnessFadeOut(wSleep,wSpeedRange?) ;OptionalForcedBlank?
  07h  SetMosaicAndSleep(wSleep,wBgFlags,wMosaicSize)
  08h  N/A (hangs)
  09h  SleepAndBlendFromCurrentToNewPalette(wSleep,vPalIndex,pNewPalette)
  0Ah  HdmaEffectsOnBg3(wSleep,wEffectType,vScrollOffset,vExtraOffset)
  0Bh  SleepWithAngleAs9wayObjShape(wSleep,pObjShape1,..,pObjShape9) ;[18A8+X]
  0Ch  DisableObjsOfAllThreads()
  0Dh  ReEnableObjsOfAllThreads()
  0Eh  SleepWithXYsignAs9wayPlayerGenderObjShape(wSleep,pObjShape1,..,Shape9)
  0Fh  N/A (hangs)
  10h  SleepAndSetXYpos(wSleep,vX,vY)
  11h  SleepAndMoveTowardsTargetXYpos(wSleep,vX,vY)
  12h  SleepAndMoveByIncomingXYstep(wSleep,vX,vY)
  13h  SleepAndMoveAndAdjustXYstep(wSleep,vRotationAngleToOldXYstepOrSo?)
  14h  SleepAndMoveWithinBoundary(wSleep,vX1,vX2,vY1,vY2,wFactor?)
  15h  SleepAndMoveChangeBothXYstepsIfCollideOtherThread(wSleep,wBounceSpeed?)
  16h  SleepAndMoveAndIncrementXYstep(wSleep,vXincr,vYincr,qXlimit,qYlimit)
  17h  SleepAndMoveByIncomingYstepAndWavingXstep(wSleep,wY)
  18h  SleepAndMoveAndAccelerateTowardsTarget(wSleep,vX,vY,vSpeed)
  19h  SleepAndMoveAndSomethingComplicated?(wSleep,vX,vY)  ;out: X,Y=modified
  1Ah  AdjustXYstep(wNewSpeedOrSo?) ;in: [18A8+X]=angle
  1Bh  MoveByOldXYstepWithoutSleep()
  1Ch  SleepAndMoveChangeXYstepIfCollideOtherThread(wSleep,vMask,vX?,vY?)
  1Dh  N/A (hangs)
  1Dh  N/A (hangs)
  1Fh  N/A (hangs)
  20h  Goto(pTarget)
  21h  Gosub(pTarget)   ;max nesting=8 (or less when also using Loops)
  22h  Return()         ;return from Gosub
  23h  QuitThread()     ;terminate thread completely
  24h  LoopStart(wRepeatCount)  ;see token 62h (LoopNext)
  25h  Sleep(wSleep)
  26h  MathsLet(vA,vB)       ;A=B
  27h  MathsAdd(vA,vB)       ;A=A+B      ;1998 if unsigned carry
  28h  MathsSub(vA,vB)       ;A=A-B      ;1998 if signed overflow
  29h  MathsAnd(vA,vB)       ;A=A AND B  ;1998 if nonzero
  2Ah  MathsOr(vA,vB)        ;A=A OR B   ;1998 if nonzero
  2Bh  MathsXor(vA,vB)       ;A=A XOR B  ;1998 if nonzero
  2Ch  MathsNot(vA)          ;A=NOT A    ;1998 if nonzero
  2Dh  MathsMulSigned(vA,vB) ;A=A*B/100h ;1998 never (tries to be overflow)
  2Eh  MathsDivSigned(vA,vB) ;A=A/B*100h ;1998 if division by 0
  2Fh  SignedCompareWithConditionalGoto(vA,wOperator,vB,pTarget)
  30h  GotoIf_1998_IsNonzero(pTarget)
  31h  GotoIf_1998_IsZero(pTarget)
  32h  GotoArray(vArrayIndex,pPointerToArrayWithTargets)
  33h  ReadJoypad(bJoypadNumber,wX,wY)
  34h  CreateAnotherInterpreterThreadWithLimit(vThreadCount,bLimit,pEntry)
  35h  CheckIfXYposCollidesWithFlaggedThreads(vFlagMask) ;out: 1998=ID
  36h  GetUnsignedRandomValue(vA,wB) ;A=Random MOD B, special on B>7FFFh
  37h  SetObjWidthDepthFlagmask(vWidth,vDepth,vMask) ;for collide checks
  38h  CreateAnotherInterpreterThreadWithIncomingXYpos(vX,vY,pEntrypoint)
  39h  N/A (hangs)
  3Ah  SoundApuMessage00h_nnh(vParameter8bit)
  3Bh  SoundApuMessage01h_nnnh(vLower6bit,bMiddle2bit,bUpper2bit)
  3Ch  SoundApuMessage02h_nnnnh(vLower6bit,bMiddle2bit,bUpper2bit)
  3Dh  SoundApuUpload(bMode,pPtrToPtrToData)
  3Eh  SetPpuBgModeKillAllOtherThreadsAndResetVariousStuff(bBgMode)
  3Fh  SetTemporaryTableForBanksF1hAndUp(vTableNumber,pTableBase)
  40h  KillAllFlaggedThreads(vMask)  ;ignores flags, and kills ALL when Mask=0
  41h  SetBUGGEDTimerHotspot(wHotspot) ;BUG: accidently ORed with AE09h
  42h  Ppu_Bg1_Bg2_SetScrollPosition(vX,vY)
  43h  Ppu_Bg1_Bg2_ApplyScrollOffsetAndSleep(wSleep,vX,vY)
```

```
  44h  NopWithDummyParameters(wUnused,wUnused)
  45h  NopWithoutParameters()
  46h  AllocateAndInitObjTilesOrUseExistingTiles(wLen,pSrc)
  47h  AllocateAndInitObjPaletteOrUseExistingPalette(pSrc)
  48h  DmaObjTilesToVram(wObjVramAddr,wOBjVramEnd,pSrc)
  49h  SetObjPalette(wObjPalIndex,wObjPalEnd,pSrc)
  4Ah  SramAddSubOrSetMoney(bAction,vLower16bit,vMiddle16bit,vUpper16bit)
  4Bh  SramUpdateChksumAndBackupCopy()
  4Ch  N/A (hangs)
  4Dh  N/A (hangs)
  4Eh  N/A (hangs)
  4Fh  N/A (hangs)
  50h  TestAndGotoIfNonzero(vA,vB,pTarget)  ;Goto if (A AND B)<>0
  51h  TestAndGotoIfZero(vA,vB,pTarget)     ;Goto if (A AND B)==0
  52h  InitNineGeneralPurposePrivateVariables(wA,wB,wC,wD,wE,wF,wG,wH,wI)
  53h  MultipleCreateThreadBySelectedTableEntries(vFlags,vLimit,pPtrToTable)
  54h  PrepareMultipleGosub()  ;required prior to token 6Ah
  55h  StrangeXYposMultiplyThenDivide(wA,wB) ;Pos=Pos*((B-A)/2)/((B-A)/2)
  56h  BuggedForceXYposIntoScreenArea() ;messes up xpos and/or hangs endless
  57h  Maths32bitAdd16bitMul100h(vA(Msw),vB) ;A(Msw:Lsw)=A(Msw:Lsw)+B*100h
  58h  Maths32bitSub16bitMul100h(vA(Msw),vB) ;A(Msw:Lsw)=A(Msw:Lsw)-B*100h
  59h  SoundApuUploadWithTimeout(wTimeout,pPtrToPtrToData)
  5Ah  N/A (hangs)
  5Bh  N/A (hangs)
  5Ch  N/A (hangs)
  5Dh  N/A (hangs)
  5Eh  N/A (hangs)
  5Fh  N/A (hangs)
  60h  CallMachineCodeFunction(pTarget)
  61h  SetTemporaryOffsetFor0AxxxxhVariables(vOffset)
  62h  LoopNext()  ;see token 24h (LoopStart)
  63h  SetForcedBlankAndSleepOnce()
  64h  ClearForcedBlankAndSleepOnce()
  65h  AllocateAndInitObjPaletteAndObjTilesOrUseExistingOnes(pSrc) ;fragile
  66h  WriteBgTiles(wBgNumber,pPtrTo16bitLenAnd24bitSrcPtr)
  67h  WritePalette(pPtrTo16bitLenAnd24bitSrcPtr)  ;to backdrop/color0 and up
  68h  WriteBgMap(wBgNumber,pPtrTo16bitLenAnd24bitSrcPtr)
  69h  KillAllOtherThreads()
  6Ah  MultipleGosubToSelectedTableEntries(vFlags,pPtrToTable) ;see token 54h
  6Bh  AllocateAndInitBgPaletteTilesAndMap2(vX1,vY1,pPtrToThreePtrs,vBgMapSize)
  6Ch  DeallocateAllObjTilesAndObjPalettes()
  6Dh  BuggedSetBgParameters(bBgNumber,pPtr,wXsiz,wYsiz,wUnused,wUnused)
  6Eh  BuggedSetUnusedParameters(bSomeNumber,pPtr,wX,wY)
  6Fh  BuggedChangeBgScrolling(wX,wY)
  70h  PauseAllOtherThreads()
  71h  UnPauseAllOtherThreads()
  72h  GosubIfAccessedByPlayer(pGosubTargetOrPeopleFolderID)
  73h  Dma16kbyteObjTilesToTempBufferAt7F4000h()   ;Backup OBJ Tiles
  74h  Dma16kbyteObjTilesFromTempBufferAt7F4000h() ;Restore OBJ Tiles
  75h  SetFixedPlayerGenderObjShape(pSrc,wLen1,wLen2)
  76h  InstallPeopleIfSatelliteIsOnline() ;create all people-threads
  77h  KillAllOtherThreadsAndGotoCrash()  ;Goto to FFh-filled ROM at 829B5Eh
  78h  ZerofillBgBufferInWram(vBgNumber)
  79h  ChangePtrToObjPriority(vVariableToBePointedTo)  ;default is <Ypos>
  7Ah  ChangeObjVsBgPriority(vPriorityBits)     ;should be (0..3 * 1000h)
  7Bh  SetXYposRelativeToParentThread(vX,vY)
  7Ch  TransferObjTilesAndObjPaletteToVram(pPtrToPtrsToPaletteAndTileInfo)
  7Dh  AllocateAndInitBgPaletteTilesAndMap1(vX1,vY1,pPtrToThreePtrs,vBgMapSize)
  7Eh  DrawMessageBoxAllAtOnce(vWindowNumber,vDelay,vX,vY,pPtrToString)
  7Fh  DrawMessageBoxCharByCharBUGGED(..)  ;works only via CALL, not token 7Fh
  80h..FFh  Reserved/Crashes (jumps to garbage function addresses)
```

Legend for Token Parameters

```
  v   16bit Global or Private Variable or Immediate (encoded as 3 token bytes)
  p   24bit Pointer (3 token bytes) (banks F0h..FFh translated, in most cases)
  b   8bit  Immediate (encoded directly as 1 token byte)
  w   16bit Immediate (encoded directly as 2 token bytes)
  q   16bit Immediate (accidently encoded as 3 token bytes, last byte unused)
```

3-byte Variable Encoding (v)

```
  +/-00nnnnh  -->  +/-nnnnh          R     ;immediate
  +/-01nnnnh  -->  +/-[nnnnh+X]      R/W   ;private variable (X=thread_id*2)
  +/-02nnnnh  -->  +/-[nnnnh]        R/W   ;global variable
  +  03nnnnh  -->  +[nnnnh+[19A4h]]  W     ;special (write-only permission)
  +  09nnnnh  -->  +[nnnnh+[19A4h]]  R/W   ;special (read/write permission)
  +  0Annnnh  -->  +[nnnnh+[19A4h]]  R     ;special (read-only permission)
  Examples: 000001h or FF0001h (aka -00FFFFh) are both meaning "+0001h".
  021111h means "+[1111h]", FDEEEF (aka -021111h) means "-[1111h]".
```

3-byte Pointer Encoding (p)

```
  00nnnnh..EFnnnnh     --> 00nnnnh..EFnnnnh      (unchanged)
  F0nnnnh              --> TokenProgramPtr+nnnn  (relative)
  F1nnnnh (or F2nnnnh) --> [[AFh+0]+nnnn*3]      (indexed by immediate)
  F3nnnnh              --> [[AFh+0]+[nnnn+X]*3]  (indexed by thread-variable)
  F4nnnnh              --> [[AFh+0]+[nnnn]*3]    (indexed by global-variable)
  F5nnnnh (or F6nnnnh) --> [[AFh+3]+nnnn*3]      (indexed by immediate)
  F7nnnnh              --> [[AFh+3]+[nnnn+X]*3]  (indexed by thread-variable)
  F8nnnnh              --> [[AFh+3]+[nnnn]*3]    (indexed by global-variable)
  F9nnnnh (or FAnnnnh) --> [[AFh+6]+nnnn*3]      (indexed by immediate)
  FBnnnnh              --> [[AFh+6]+[nnnn+X]*3]  (indexed by thread-variable)
  FCnnnnh              --> [[AFh+6]+[nnnn]*3]    (indexed by global-variable)
  FDnnnnh..FFnnnnh     --> crashes               (undefined/reserved)
```

2-byte Operators for Signed Compare (Token 2Fh)

```
  0000h Goto_if_less              ;A<B
  0001h Goto_if_less_or_equal     ;A<=B
  0002h Goto_if_equal             ;A=B
  0003h Goto_if_not_equal         ;A<>B
  0004h Goto_if_greater           ;A>B
  0005h Goto_if_greater_or_equal  ;A>=B
```

ControlSubThread(pEntrypoint) values

```
  xx0000h Pause
  xx0001h UnpauseSubThreadAndReenableObj
  xx0002h PauseAfterNextFrame
  xx0003h PauseAndDisableObj
```

```
  xx0004h ResetAndRestartSubThread
  xx0005h KillSubThread
  NNNNNNh Entrypoint (with automatic reset; only if other than old entrypoint)
```
Maximum Stack Nesting is 4 Levels (Stack is used by Gosub and Loop tokens).

Token Extensions (some predefined functions with Token-style parameters)
These are invoked via CALL Token (60h), call address, followed by params.
```
  809225h CallKillAllMachineCodeThreads()
  80B47Dh CallGetTextLayerVramBase()
  80B91Eh CallClearBg3TextLayer()
  818EF9h CallSetApuRelatedPtr()
  818F06h CallDrawMessageBoxCharByChar(vWindowNumber,vDelay,vX,vY,pPtrToString)
  818FF0h CallDrawBlackCircleInLowerRightOfWindow()
  81903Dh CallDisplayButton_A_ObjInLowerRightOfWindow()
  81A508h CallSetGuiBorderScheme(pAddr1,pAddr2)
  81A551h CallSetTextWindowBoundaries(wWindowNumber,bXpos,bYpos,bXsiz,bYsiz)
  81A56Eh CallHideTextWindow(wWindowNumber)
  81A57Bh CallSelectWindowBorder(wWindowNumber,wBorder) ;0..3, or FFh=NoBorder
  81A59Ah CallSelectTextColor(wWindowNumber,bColor,bTileBank,bPalette)
  81A5C3h CallClearTextWindowDrawBorder(wWindowNumber)
  81A5D2h CallZoomInTextWindow(wWindowNumber,wZoomType)  ;\1,2,3=Zoom HV,V,H
  81A603h CallZoomOutTextWindow(wWindowNumber,wZoomType) ;/0=None/BuggyWinDiv2
  81A634h CallSetGuiColorScheme(pAddr)
  81A65Dh CallChangePaletteOfTextRow(vX,vY,vWidth,vPalette)
  81A693h CallPeekMemory16bit(vDest,pSource)
  81A6B4h CallPokeMemory16bit(vSource,pDest)
  81C7D0h CallInitializeAndDeallocateAllObjTilesAndObjPalettes()
  81C871h CallDeallocateAllObjs()
  81CDF9h CallBackupObjPalette()
  81CE09h CallRestoreObjPalette()
  829699h CallUploadPaletteVram(pSource,wVramAddr,bPaletteIndex)
  88932Fh CallTestIfFolderExists()  ;in: 0780, out: 1998,077C,077E
  88D076h CallTestIfDoor()
  99D9A4h CallSelectPlayerAsSecondaryThread  ;[19A4]=PlayerThreadId*2
```
Note: Some of these Call addresses are also listed in a 24bit-pointer table at address 808000h (though the (BIOS-)code uses direct 8xxxxxh values instead of indirect [808xxxh] values).

Token Functions (some predefined token-functions)
These can be invoked with GOSUB token (or GOTO or used as thread entrypoint):
```
  99D69A EnterTown (use via goto, or use as entrypoint)
  828230 DeallocMostBgPalettesAndBgTiles ;except tile 000h and color 00h-1Fh
  88C1C6 SetCursorShape0
  88C1D0 SetCursorShape1
  88C1E0 SetCursorShape2
  88C1EA SetCursorShape3
  88C1F4 SetCursorShape4
  88C1FE SetCursorShape5
  99D8AB PauseSubThreadIfXYstepIsZero
  99D8CD MoveWithinX1andX2boundaries
  99D903 MoveWithinY1andY2boundaries
```
Note: Some of the above functions are also listed in the table at 808000h.

Compressed Data
Some of the Functions can (optionally) use compressed Tile/Map/Palette data.
SNES Decompression Formats
Note: The actual compressed data is usually preceeded-by or bundled-with compression flags, length entry, and/or (in-)direct src/dest pointers (that "header" varies from function to function).

SNES Cart Satellaview Chipsets

BSC-1A5B9P-01 (1995) (BIOS cartridge PCB)
```
  U1  44pin  MCC-BSC LR39197 Nintendo
  U2  36pin  ROM (36pin/40pin possible)
  U3  32pin  658512LFP-85 (4Mbit PSRAM)
  U4  28pin  LH52B256NB-10PLL (256Kbit SRAM)
  U5  8pin   MM1134 (battery controller for SRAM)
  BT1 2pin   Battery
  CN1 62pin  SNES Cartridge Edge (pin 2,33 used) (REFRESH wired to EXPAND?)
  CN2 62pin  Flash Cartridge Connector (male?)
```
There's no CIC chip (either it's contained in the MCC-chip... or in the flash card, but in that case the thing won't work without flash card?)

MAIN-BSA-01 (1995) (receiver unit/expansion port PCB)
```
  U1 20pin  74LS541 8-bit 3-state buffer/line driver
  U2 20pin  74LS541 8-bit 3-state buffer/line driver
  U3 20pin  74LS245 8-bit 3-state bus transceiver
  U4 8pin   SPR-BSA (unknown, might be controlled via port 2198h or 2199h?)
  U5 100pin DCD-BSA (custom Nintendo chip)
  U6 64pin  MN88821 (maybe a MN88831 variant: Satellite Audio Decoder)
  U7 18pin  AN3915S Clock Regenerator (for amplifying/stabilizing Y1 crystal)
  U8 4pin   PQ05RH1L (5V regulator with ON/OFF control)
  U9 14pin  LM324 Quad Amplifier
  Y1 2pin   18.432MHz crystal
  T1 4pin   ZJYS5102-2PT Transformator
  T2 4pin   ZJYS5102-2PT Transformator
  CN1 28pin SNES Expansion Port
  CN2 38pin Expansion Port (EXT) (believed to be for modem)
  CN3 3pin  To POWER and ACCESS LEDs on Front Panel
  CN4 7pin  Rear connector (satellite and power supply?)
```

BSMC-AF-01 (Memory Card PCB) (to be plugged into BIOS cartridge)
```
  U1  54pin FLASH chip (unknown manufacturer and part number?)
  CN1 62pin Flash Cartridge Connector (female?)
```
There are no other chips on this PCB (only capacitors and resistors).

BSC-1A5M-01 (1995) (GAME cartridge with onboard FLASH cartridge slot)

```
U1  36pin  ROM
U2  28pin  SRAM (32Kbytes)
U3  16pin  MAD-1A
U4  16pin  CIC D411B
BT1 2pin   Battery CR2032
CN1 62pin  SNES Cartridge Edge (pin 2,33 used) (REFRESH wired to EXPAND?)
CN2 62pin  Flash Cartridge Connector (male 2x31 pins)
```
Used by "Derby Stallion 96" (and maybe other games, too).

BSC-1L3B-01 (1996) (GAME cartridge with SA1 and onboard FLASH cartridge slot)
```
U1  44pin  ROM
U2  28pin  SRAM (8Kbytes)
U3  128pin SA1
U4  8pin   MM1026AF (battery controller for SRAM)
BT1 2pin   Battery
CN1 62pin  SNES Cartridge Edge (pin 2,33 used) (REFRESH wired to EXPAND?)
CN2 62pin  Flash Cartridge Connector (male?)
```
Used by "Itoi Shigesato no Bass Tsuri No. 1" (and maybe other games, too).

SNES Cart Data Pack Slots (satellaview-like mini-cartridge slot)

Data Packs
Data Packs are Satellaview 8M Memory Packs which have data meant to be used as expansion for a Data Pack-compatible game. Data Pack-compatible game cartridges have a resemblence to the BS-X Cartridge itself.

Usage
For most of these games, Data was distributed via St.GIGA's Satellaview services. Same Game and SD Gundam G-Next had some Data Packs sold as retail in stores. RPG Tsukuru 2, Sound Novel Tsukuru and Ongaku Tsukuru Kanaderu could save user-created data to 8M Memory Packs.

Cartridges with Data Pack Slot
```
Derby Stallion 96                 (SpecialLoROM, 3MB ROM, 32K RAM)
Itoi Shigesato no Bass Tsuri No. 1 (SA-1, map-able 4MB ROM, 8K RAM)
Joushou Mahjong Tenpai            (HiROM, 1MB)
Ongaku Tukool/Tsukuru Kanaderu    (HiROM, 1MB)
RPG Tukool/Tsukuru 2              (LoROM, 2MB)
Same Game Tsume Game              (HiROM, 1MB)
Satellaview BS-X BIOS             (MCC, 1MB ROM) (FLASH at C00000h)
SD Gundam G-NEXT                  (SA-1, map-able 1.5MB ROM, 32K RAM)
Sound Novel Tukool/Tsukuru        (SpecialLoROM, 3MB ROM, 64K RAM)
```
Aside from the BS-X BIOS, two of the above games are also accessing BS-X hardware via I/O Ports 2188h/2194h/etc (Derby Stallion 96, Itoi Shigesato no Bass Tsuri No. 1). For doing that, the cartridges do probably require the EXPAND pin to be wired properly (ie. wired to REFRESH or so?).

Cartridge Header of cartridges with Data Pack Slot
The presence of Data Pack Slots is indicated by a "Z" as first letter of Game Code:
```
[FFB2h]="Z"   ;first letter of game code
[FFB5h]<>20h  ;game code must be 4-letters (not space padded 2-letters)
[FFDAh]=33h   ;game code must exist (ie. extended header must be present)
```

Data Pack Mapping
```
MCC (BSX-BIOS)        FLASH at C00000h (continous) (mappable via MCC chip)
SA-1                  FLASH at <unknown address> (probably mappable via SA1)
HiROM                 FLASH at E00000h (probably continous)
LoROM/SpecialLoROM    FLASH at C00000h (looks like 32K chunks)
```

LoROM/SpecialLoROM Mapping Notes
The FLASH memory seems to be divided into 32K chunks (mirrored to Cn0000h and Cn8000h) (of which, Derby Stallion 96 uses Cn8000h, RPG Tukool uses Cn0000h, and Ongaku Tukool uses both Cn0000h and Cn8000h).
The two 3MB SpecialLoROM games also have the ROM mapped in an unconventional fashion:
```
1st 1MB of ROM mapped to banks 00-1F
2nd 1MB of ROM mapped to banks 20-3F and A0-BF
3rd 1MB of ROM mapped to banks 80-9F
1MB of Data Pack FLASH mapped to banks C0-DF
32K..64K SRAM mapped to banks 70-71
```
Despite of memory-mirroring of 2nd MB, the checksum-mirroring goes on 3rd MB?
Note: Above mapping differs from "normal" 3MB LoROM games like Wizardry 6 (which have 3rd 1MB in banks 40h-5Fh).

SNES Cart Nintendo Power (flashcard)

Nintendo Power cartridges are official FLASH cartridges from Nintendo (released only in Japan). Unlike the older Satellaview FLASH cartridges, they do connect directly to the SNES cartridge slot. The capacity is 4MByte FLASH and 32KByte battery-backed SRAM.

FLASH (512Kbyte blocks)
The FLASH is divided into eight 512Kbyte blocks. The first block does usually contain a Game Selection Menu, the other blocks can contain up to seven 512KByte games, or other combinations like one 3MByte game and one 512KByte game. Alternately, the cartridge can contain a single 4MByte game (in that case, without the Menu).

SRAM (2Kbyte blocks) (battery-backed)
The SRAM is divided into sixteen 2Kbyte blocks for storing game positions. Games can use one or more (or all) of these blocks (the menu doesn't use any of that memory).

SNES Cart Nintendo Power - I/O Ports
SNES Cart Nintendo Power - Directory

Nintendo Power Games
Games have been available at kiosks with FLASH Programming Stations. There are around 150 Nintendo Power games: around 21 games exclusively released only for Nintendo Power users, and around 130 games which have been previously released as normal ROM cartridges.

Nintendo Power PCB "SHVC-MMS-02" (1997) Chipset

```
U1  18pin CIC       ("F411B Nintendo")
U2 100pin MX15001   ("Mega Chips MX15001TFC")
U3  44pin 16M FLASH ("MX 29F1601MC-11C3") (2Mbyte FLASH)
U4  44pin 16M FLASH ("MX 29F1601MC-11C3") (2Mbyte FLASH)
U5  44pin 16M FLASH (N/A, not installed)
U6  28pin SRAM      ("SEC KM62256CLG-7L") (32Kbyte SRAM)
U7   8pin MM1134    ("M 707 134B") (battery controller)
BAT1 2pin Battery   ("Panasonic CR2032 +3V")
```

Nintendo Power Menu Cartridge Header

```
Gamecode:        "MENU" (this somewhat indicates the "MX15001" chip)
ROM Size:        512K (the menu size, not including the other FLASH blocks)
SRAM Size:       0K (though there is 32Kbyte SRAM for use by the games)
Battery Present: Yes
Checksum:        Across 512Kbyte menu, with Directory assumed to be
                 FFh-filled (except for the "MULTICASSETTE 32" part)
```

The PCB doesn't contain a ROM (the Menu is stored in FLASH, too).

Nintendo Power Menu Content

```
ROM Offset  SNES Address Size   Content
000000h     808000h      4xxxh  Menu Code (around 16K, depending on version)
004xxxh     80xxxxh      3xxxh  Unused (FFh-filled)
007FB0h     80FFB0h      50h    Cartridge Header
008000h     818000h      40000h Unused (FFh-filled)
048000h     898000h      372Bh  Something (APU code/data or so)
04B72Bh     8xxxxxh      47D5h  Unused (FFh-filled)
050000h     8A8000h      8665h  Something (VRAM data or so)
058665h     8Bxxxxh      798Bh  Unused (FFh-filled)
060000h     8C8000h      10000h Directory (File 0..7) (2000h bytes/entry)
070000h     8E8000h      10000h Unused (FFh-filled)
```

Note

Nintendo has used the name "Nintendo Power" has been used for various things:

```
Super Famicom Flashcards (in Japan)
Gameboy Color Flashcards (in Japan)
Super Famicom Magazine (online via Satellaview BS-X) (in Japan)
Official SNES Magazine (printout) (in USA)
```

SNES Cart Nintendo Power - I/O Ports

Nintendo Power I/O Ports

The I/O ports at 002400h-002401h are used for mapping a selected game. Done as follows:

```
mov  [002400h],09h
cmp  [002400h],7Dh
jne  $  ;lockup if invalid
mov  [002401h],28h
mov  [002401h],84h
mov  [002400h],06h
mov  [002400h],39h
mov  [002400h],80h+(Directory[n*2000h+0] AND 0Fh)
jmp  $  ;lockup (until reset applies)
```

After the last write,the MX15001 chip maps the desired file, and does then inject a /RESET pulse to the SNES console, which resets the CPU, APU (both SPC and DSP), WRAM (address register), and any Expansion Port hardware (like Satellaview), or piggyback cartridges (like Xband modem). The two PPU chips and the CIC chip aren't affected by the /RESET signal. The overall effect is that it boots the selected file via it's Reset vector at [FFFCh].
Note that the written value is the file index (ie. 80h+02h for the 2nd file), not a memory block number. So the MX15001 must somewhere look-up directory information for actually mapping the corresponding FLASH and SRAM blocks.

FLASH Base Address

The Block Number of the first FLASH block is stored at [0001h] in the Directory. The MX15001 chip might use the entry for mapping FLASH, unless the MX15001 contains its own rewrite-able directory memory (?)

SRAM Base Address (Unknown)

As by now, nobody does seem to have ever dumped Nintendo Power directory entries for games that do contain SRAM. So far it's totally unknown if there is a "First SRAM Block" entry in the directory (if it is there, then it'd be probably located in some bits at [0000h..0002h]). If there is no entry, then one could probably calculate it by summing-up the SRAM size entries from the <preceeding> Directory Entries.
As for the actual SRAM mapping, the MX15001 may look-up the FLASH directory, or its own rewrite-able directory memory (in case such memory does exist).

SRAM/FLASH Limits (Unknown)

Unknow if the hardware does apply any mapping limits (so that only the blocks used by a specfic game are mapped into memory).

LoROM Mapping (Unknown)

Most games (and the Menu) are using LoROM mapping, with up to 2MB in 64 banks of 32K in bank 00h-3Fh and mirrors in bank 80h-BFh. So far it's simple. The unknown part is that there are also 4MB games, that exceed the above 64 banks; the extra memory is probably somehow mapped in the "HiROM" area in banks 40h-7Dh and C0h-FFh. However, details are unknown.

HiROM Mapping (Unknown)

The Menu, and games like Contra and Parodius are using LoROM mapping. Games like Rockman 7 are using HiROM mapping. The Directory doesn't NOT contain any HiROM flag (and in case of 4MB games, there isn't a Directory at all). So, it's unclear how the hardware can know whether to map LoROM or HiROM, possible ways would be:

```
- Upper/Lower 32K are exchanged in 64K HiROM banks (vectors always at 7Fxxh)
- Hardware examines headers at 7Fxxh and FFxxh in selected 512K block
- The MX15001 chips contains rewrite-able memory (with more directory info)
```

There are also

FLASH Chip Programming (Unknown)

The Bootmenu only allows file-selection, but doesn't contain any programming functions. The games do probably (?) only use SRAM as storage, and thus don't contain any FLASH programming functions either. So, programming functions seem to exist only in the official programming devices (located in shops).
The "MX 29F1601MC-11C3" chip is probably a variant of the 29F1610, 29F1611, 29F1615 chips from Macronix (the chip's part number really ends with "01", not with "10", "11", nor "15"), if so, then it should be programmed by writing commands to address xx5555h and xx2AAAh, and the erase sector size would be 128Kbytes). Though Nintendo may have also installed different FLASH chips from other makers and with other protocols in some cartridges(?)
To prevent piracy, it's quite likely that the MX15001 prevents writes to FLASH memory unless it has received some special unlocking command, there may be

also special (SNES-incompatible) cartridge slot pin-outs required for writing; unknown if any such protection does exist & how it does work.

SNES Cart Nintendo Power - Directory

Directory Area

```
ROM Offset  SNES Address Size   Content
060000h     8C8000h      2000h  File 0 (Menu)
062000h     8CA000h      2000h  File 1
064000h     8CC000h      2000h  File 2
066000h     8CE000h      2000h  File 3
068000h     8D8000h      2000h  File 4
06A000h     8DA000h      2000h  File 5
06C000h     8DC000h      2000h  File 6
06E000h     8DE000h      2000h  File 7
070000h     8E8000h      10000h Unused (FFh-filled)
```

The last 64Kbyte are probably usable as further file entries in cartridges bigger than 4Mbyte (the Menu software in the existing cartridges is hardcoded to process only files 1..7) (whilst Port 2400h seems to accept 4bit file numbers).

Directory Entry Format

```
0000h 1    Directory index (00h..07h for Entry 0..7) (or FFh=Unused Entry)
0001h 1    First 512K-FLASH block (00h..07h for block 0..7)
0002h 1    Unknown (usually 00h) maybe First ??K-SRAM block and/or Hirom flag
0003h 2    Number of 512K-FLASH blocks (mul 4) (=0004h..001Ch for 1..7 blks)
0005h 2    Number of 2K-SRAM blocks (mul 16)   (=0000h..0100h for 0..16 blks)
0007h 12   Gamecode (eg. "SHVC-MENU-  ", "SHVC-AGPJ-  ", or "SHVC-CS  -  ")
0013h 44   Title in Shift-JIS format (padded with 00h's) (not used by Menu)
003Fh 384  Title Bitmap (192x12 pixels, in 30h*8 bytes, ie. 180h bytes)
01BFh 10   Date "MM/DD/YYYY"
01C9h 8    Time "HH:MM:SS"
01D1h 8    Law  "LAWnnnnn" (eg. "LAW01712")
01D9h 7703 Unused (1E17h bytes, FFh-filled)
1FF0h 16   For File0: "MULTICASSETTE 32" / For Files 1-7: Unused (FFh-filled)
```

Directory Index

Directory Index indicates if the Entry is used (FFh=unused). If it is used, then it must be equal to the current directory entry number (ie. a rather redundant thing, where the index is indexing itself). The lower 4bit of the index value is written to Port 2400h, which is probably causing the MX15001 chip to load the First FLASH/SRAM block numbers from the corresponding directory entry, and to apply them as mapping base addresses.

First FLASH/SRAM Block

The First FLASH block number is stored in lower some bits of [0001h]. There is probably also an entry for the First SRAM block number, possibly in whatever bits at [0001h] or [0002h]? The directory doesn't contain any flag that indicates HiROM or LoROM mapping.
The Menu doesn't use these entries, instead, they are probably automatically deciphered by the MX15001 chip. [This isn't verified. Another theory found in internet claims that the Block-numbers are stored in additional FLASH memory contained in the MX15001 chip - however, that theory isn't verfied either.]
There seems to be no support for fragmented FLASH/SRAM files (ie. the programming station must probably erase & rewrite the entire cartridge, with the old used/unused blocks re-ordered so that they do form continous memory blocks).

Number of FLASH/SRAM Blocks (displayed in Menu)

These entries are used to display the amount of used/unused blocks. Free FLASH blocks are shown as blue "F" symbols, free SRAM blocks as red "B" symbols, used blocks as gray "F" and "B" symbols. The menu doesn't show <which> game uses how many blocks.

Title Bitmap (displayed in Menu)

The 192x12 pixel title bitmap is divided into eight 24x12 pixel sections, using a most bizarre encoding: Each section is 30h bytes in size (enough for 32 pixel width, but of these 32 pixels, the "middle" 4 pixels are overlapping each other, and the "right-most" 4 pixels are unused. The byte/pixel order for 12 rows (y=0..11) is:

```
Left 8 pixels   = (Byte[00h+y*2])
Middle 8 pixels = (Byte[01h+y*2]) OR (Byte[18h+y*2] SHR 4)
Right 8 pixels  = (Byte[18h+y*2] SHL 4) OR (Byte[19h+y*2] SHR 4)
```

The result is displayed as a normal 192-pixel bitmap (without any spacing between the 24-pixel sections). The bits in the separate bytes are bit7=left, bit0=right. Color depth is 1bit (0=dark/background, 1=bright/text). The bitmap does usually contain japanese text without "special" features, though it could also be used for small icons, symbols, bold text, greek text, etc.

Text Fields (not used by Menu)

The Shift-JIS Title, and ASCII Game Code, Date, Time, Law & Multicassete strings aren't used by the Menu. The 5-digit Law number is usually (always?) same for all files on the cartridge, might be the serial number of the cartridge, or of the station that has programmed it. The Multicassette number does probably indicate the FLASH size in MBits (it's always 32 for the existing 32Mbit/4MByte cartridges).

SNES Cart Sufami Turbo (Mini Cartridge Adaptor)

The Sufami Turbo from Bandai is an adaptor for low-cost mini-cartridges. Aside from cost-reduction, one special feature is that one can connect two cartridges at ones (so two games could share ROM or SRAM data). The BIOS in the adaptor provides a huge character set, which may allow to reduce ROM size of the games.

SNES Cart Sufami Turbo General Notes
SNES Cart Sufami Turbo ROM/RAM Headers
SNES Cart Sufami Turbo BIOS Functions & Charset

SNES Cart Sufami Turbo General Notes

Sufami Turbo Hardware

The "adaptor" connects to the SNES cartridge socket, it contains the BIOS ROM, and two slots for "mini-carts". Slot A for the game being played, Slot B can contain another game (some games include features that allow to access game position data from other games, some may also access ROM data from other games).

Sufami Turbo Memory Map

```
00-1F:8000h-FFFFh  BIOS ROM (always 256Kbytes)            (max 1MByte)
20-3F:8000h-FFFFh  Cartridge A ROM (usually 512Kbytes)    (max 1MByte)
```

```
60-63:8000h-FFFFh  Cartridge A SRAM (usually 0/2/8 Kbytes) (max 128Kbyte)
70-73:8000h-FFFFh  Cartridge B SRAM (usually 0/2/8 Kbytes) (max 128Kbyte)
80-FF:8000h-FFFFh  Mirror of above banks
```

Memory Notes
The BIOS detects max 128Kbyte (64 pages) SRAM per slot, some games are (maybe accidently) exceeding that limit (eg. Poi Poi Ninja zerofills 256 pages).
Some games (eg. SD Gundam Part 1) access SRAM slot B at 700000h rather than 708000h.
Some games (eg. SD Ultra Battle) may fail if the SRAM in slot B is uninitialized (ie. before linking games in Slot A and B, first launch them separately in Slot A).
When not using BIOS functions, one can destroy all WRAM locations, except for WRAM[00000h] (which MUST be nonzero to enable the Game NMI handler & disable the BIOS NMI handler).

Sufami Turbo ROM Images
The games are typically 512Kbyte or 1MByte in size. Existing ROM-Images are often 1.5Mbytes or 2MBytes - those files do include the 256KByte BIOS-ROM (banks 00h-07h), plus three mirrors of the BIOS (banks 08h-1Fh), followed by the actual 512Kbyte or 1MByte Game ROM (bank 20h-2Fh or 20h-3Fh).
There are also a few 3MByte ROM-images, with additional mirrors of the game (bank 30h-3Fh), followed by a second game (bank 40h-4Fh), followed by mirrors of the second game (bank 50h-5Fh).
That formats are simple (but very bloated) solutions to load the BIOS & Game(s) as a "normal" LoROM file.

Sufami Turbo Games
There have been only 13 games released:

```
Crayon Shin Chan
Gegege No Kitarou
Gekisou Sentai Car Ranger
Poi Poi Ninja                    ;-link-able with itself (2-player sram)
Sailor Moon Stars Panic 2
SD Gundam Generations: part 1    ;\
SD Gundam Generations: part 2    ;
SD Gundam Generations: part 3    ; link-able with each other
SD Gundam Generations: part 4    ;
SD Gundam Generations: part 5    ;
SD Gundam Generations: part 6    ;/
SD Ultra Battle: Seven Legend    ;\link-able with each other
SD Ultra Battle: Ultraman Legend ;/
```

All of them available only in Japan, released between June & September 1996. Thereafter, the games may have been kept available for a while, but altogether, it doesn't seem to have been a too successful product.

Component List for Sufami Turbo Adaptor
PCB "SHVC TURBO, BASE CASSETTE, BANDAI, PT-923"

```
IC1   18pin  unknown (CIC)
IC2   16pin  "74AC139" or so
IC3   40pin  SUFAMI TURBO "LH5326NJ" or so (BIOS ROM) (256Kbyte)
IC4   8pin   unknown
CP1   unknown (flashlight? oscillator? strange capacitor?)
CN1   62pin  SNES cartridge edge (male)
CN2   40pin  Sufami Cartridge Slot A (Game to be played)
CN3   40pin  Sufami Cartridge Slot B (Other game to be "linked")
C1..4  2pin  capacitors for IC1..4
R1..4  2pin  resistors for unknown purpose
```

Note: Of the 62pin cartridge edge, only 43 pins are actually connected (the middle 46 pins, excluding Pin 40,48,57, aka A15/A23/SYSCK).

Component Lists for Sufami Turbo Game Carts
All unknown. Probably contains only ROM, and (optionally) SRAM and Battery. Physical SRAM size(s) are unknown (ie. unknown if there is enough memory for more than one file). Cartridge slot pin-outs are unknown.

SNES Cart Sufami Turbo ROM/RAM Headers

Sufami Turbo BIOS ROM Header
The BIOS has a rather incomplete Nintendo-like header at ROM Offset 07FB0h (mapped to 00FFB0h):

```
FFB0h Maker Code "B2"                      ;\extended header, present
FFB2h Game Code "A9PJ"                     ; even though [FFDAh]<>33h
FFB6h Reserved (10x00h)                    ;/
FFC0h Title "ADD-ON BASE CASSETE  " (really mis-spelled, with only one "T")
FFD4h Mapmode (always 30h = Fast LoROM)
FFD5h Reserved (6x00h) (no ROM/RAM size entries, no ext.header-flag, etc.)
FFDCh Dummy "checksum" value (always FFh,FFh,00h,00h)
FFE0h Exception Vectors (IRQ,NMI,Entrypoint,etc.)
```

And, there is a header-like data field at ROM-Offset 00000h (mapped to 808000h) (this part isn't really a header, but rather contains ID strings that are used by the BIOS, for comparing them with Game ROM/SRAM):

```
8000h 16 "BANDAI SFC-ADX",0,0   ;Game ROM ID
8010h 16 "SFC-ADX BACKUP",0,0   ;Game SRAM ID
```

Sufami Turbo Game ROM Header (40h bytes)
Located at ROM Offset 00000h (mapped to 208000h/408000h for Slot A/B):

```
00h 14 ID "BANDAI SFC-ADX" (required, compared against 14-byte ID in BIOS)
0Eh 2  Zero-filled
10h 14 Title, padded with spaces (can be 7bit ASCII and 8bit Japanese)
1Eh 2  Zero-filled
20h 2  Entrypoint (in bank 20h) ;game starts here (if it is in Slot A)
22h 2  NMI Vector (in bank 20h) ;if RAM[000000h]=00h: use BIOS NMI handler
24h 2  IRQ Vector (in bank 20h)
26h 2  COP Vector (in bank 20h)
28h 2  BRK Vector (in bank 20h)
2Ah 2  ABT Vector (in bank 20h)
2Ch 4  Zero-filled
30h 3  Unique 24bit ID of a Game (or series of games) (usually 0xh,00h,0yh)
33h 1  Index within a series (01h and up) (eg. 01h..06h for Gundam 1-6)
34h 1  ROM Speed (00h=Slow/2.68Mhz, 01h=Fast=3.58MHz)
35h 1  Chipset/Features (00h=Simple, 01h=SRAM or Linkable?, 03h=Special?)
36h 1  ROM Size in 128Kbyte Units (04h=512K, 08h=1024K)
37h 1  SRAM Size in 2Kbyte Units (00h=None, 01h=2K, 04h=8K)
38h 8  Zero-filled
```

Some games have additional 64 header-like bytes at ROM Offset 40h..7Fh

```
40h 1  Program code/data in some carts, 00h or 01h in other carts
```

```
  41h 63 Program code/data in some carts, 00h-filled in other carts
```
The game cartridges don't use/need a Nintendo-like header at 7Fxxh/FFxxh, but some games like SDBATTLE SEVEN do have one.

Sufami Turbo SRAM File Header (30h bytes)
```
  0000h 15 ID "SFC-ADX BACKUP",0   ;Other = begin of free memory
  000Fh 1  Zero
  0010h 14 Title (same as 0010h..001Dh in ROM Header)
  001Eh 1  Zero
  001Fh 1  Zero (except, 01h in Poi Poi Ninja)
  0020h 4  Unique ID and Index in Series (same as 0030h..0033h in ROM Header)
  0024h 1  Filesize (in 2Kbyte units)    (same as 0037h in ROM Header)
  0025h 11 Zero-filled
```
The BIOS file-functions are only reading entry 0000h (ID) and 0024h (Filesize), the BIOS doesn't write anything, all IDs and values must be filled-in by the game.
SRAM is organized so that used 2Kbyte pages are at lower addresses, free pages at higher addresses (deleting a file in the middle will relocate any pages at higher addresses). Accordingly, files are always consisting of unfragmented continous page numbers (leaving apart that there are 32Kbyte gaps in the memory map).

SNES Cart Sufami Turbo BIOS Functions & Charset

Sufami Turbo BIOS Function Summary
BIOS Function vectors (jmp 80xxxxh opcodes) are located at 80FF00h..80FF3Bh,
(the first 12 (of the 15) functions are also duplicated at 80FF80h..80FFAFh).
```
  80FF00  FillSramPages  ;in: AL=num, AH=slot, XL=first, [09h]=fillword
  80FF04  CopySramToSram ;in: AL=num, AH=direction, X/Y=first (slot A/B)
  80FF08  CopySramToWram ;in: AL=num, AH=direction, X=first, Y=slot, [09h]=addr
  80FF0C  GetChar2bpp    ;in: A=char(0000h..0FFFh), [06h]=dest_addr (64 bytes)
  80FF10  GetChar4bpp    ;in: A=char(0000h..0FFFh), [06h]=dest_addr (128 bytes)
  80FF14  GetCartType    ;out: AL/AH=Types for Slot A/B, b0=ROM, b1=SRAM, b2=?
  80FF18  GetSramSize    ;out: AL/AH=Sizes for Slot A/B, 0-4=0,2,8,32,128Kbyte
  80FF1C  FindFreeSram   ;in: AL=slot, out: AL=first_free_page, FFh=none
  80FF20  GetSramAddrTo6 ;in: AL=slot, XL=page, out: [06h]=addr
  80FF24  GetSramAddrTo9 ;in: AL=slot, XL=page, out: [09h]=addr
  80FF28  ShowHelpSwap   ;display instructions how to exchange cartridges
  80FF2C  ShowHelpNoSwap ;display instructions how to remove cartridges
  80FF30  DeleteFile     ;in: AL=first, AH=slot
  80FF34  TestSramId     ;in: AL=page, AH=slot, out: CY: 0=Used, 1=Free
  80FF38  SramToSramCopy ;in: AL=num, X=src, Y=dst; XH/YH=slot, XL/YL=first
```
Whereas,
```
  num = number of 2Kbyte pages
  slot = slot (0 or 1 for slot A or B)
  first = first 2Kbyte page number (of a file/area) (within selected slot)
  page = single 2Kbyte page number (within selected slot)
  addr = 24bit SNES memory address
  AL/AH, XL/XH, YL/YH = LSB/MSB of A,X,Y registers
```
The BIOS functions use first 16 bytes in WRAM [0000h..000Fh] for parameters, return values, and internal workspace; when using BIOS functions, don't use that memory for other purposes. [0000h] is NMI mode, don't change that even when NOT using BIOS functions.

File/SRAM Functions
These functions may be (not very) helpful for managing SRAM, they are extremly incomplete, there are no functions for creating files, or for searching specific files. See the "Header" chapter for details on SRAM headers (again, the BIOS doesn't create any headers or IDs, the game must fill-in all IDs, Titles, and other values on its own).

Character Set
The BIOS ROM contains 4096 characters (each 16x16 pixel, aka 2x2 tiles). The characters are stored at 1bit color depth in banks 04h..07h, offset 8000h-FFFFh (20h bytes/character). The GetChar2bpp and GetChat4bpp functions can be used to copy a selected character to WRAM, with bits in plane0, and the other plane(s) zerofilled.

Help Functions
The two help functions are showing some endless repeated japanese instructions about how to use, insert, remove, and exchange cartridges (similar to the instructions shown when booting the BIOS without Game cartridges inserted). If you have uploaded code to the APU, be sure to return control to the APU boot-rom, otherwise the help functions will hang.

SNES Cart X-Band (2400 baud Modem)

The X-Band is a 2400 baud modem from Catapult Entertainment Inc., licensed by Nintendo, originally released 1994 in USA, and 199x? in Japan. Aside from the SNES version, there have been also Genesis and Saturn versions.

SNES Cart X-Band Misc
SNES Cart X-Band Smart Card Reader
SNES Cart X-Band Rockwell Ports
SNES Cart X-Band Rockwell Notes
SNES Controllers X-Band Keyboard

Note
There's also another modem (which connects to controller port):
SNES Add-On SFC Modem (for JRA PAT)

SNES Cart X-Band Misc

Info...
It was used for networked gaming via phone lines.
The Xband worked by sending controller instructions, by intercepting code from the game, and patching it with its own instructions, much like the Game Genie works. (that are, probably, two separate features messed into one sentence?)
The system worked by dialing up the main server, which was located in Cupertino, California (USA), and somewhere else (Japan). The server then sent the Xband newsletters (called Bandwidth and Xband News). It also sent any patches that were needed. You could then search for opponents.

Unknown Features
There seems to be no CIC chip, so the BIOS does likewise work only with another SNES cart connected.
There is switch, for whatever on/off/mode selection. There are three LEDs for whatever purpose. And, there is some kind of a credit-card (or so) reader.

Memory Map
```
  D00000h-DFFFFFh  1MB ROM (executed here, not at C00000h-CFFFFFh)
  E00000h-E0FFFFh  64K SRAM (in two 32Kx8 chips) (unknown if BOTH have battery)
  FBC000h-FBC17Fh  I/O Ports (unknown functions?)
  FBC180h-FBC1BFh  I/O Ports (Rockwell Modem Chip)
  FBFC02h          I/O Port  (unknown functions?)
  FBFE00h          I/O Port  (unknown functions?)
  FFC000h          I/O Port  (unknown functions?)
  004F02h          I/O Port  (unknown functions?)
  00F000h          Dummy/strobe read?
  00FFE0h          Dummy/strobe read?
```
I/O Ports seem to be 8bit-wide / word-aligned (ie. one can use 8bit or 16bit writes, with the MSB ignored in the latter case). Normally ONLY the even addresses are used (some exceptions are: 8bit write 00h to FBC153h, 16bit write 0000h to FBC160h).
Some of the I/O ports outside of the FBCxxxh region might belong to other hardware? (eg. the X-Band might automatically disable any Game Genie BIOS in order to access the Game ROM).

Unknown 100pin Chip
Unknown. Probably controls the cart reader, the cheat/patching feature, and maybe also memory & I/O mapping of the other chips.

Games supported by the X-Band modem
```
  Doom
  Ken Griffey Jr. Baseball
  Killer Instinct
  Madden NFL '95
  Madden NFL '96
  Mortal Kombat II
  Mortal Kombat 3
  NBA Jam TE
  NHL '95
  NHL '96
  Super Mario Kart
  Weaponlord
```
and,
```
 Kirby's Avalanche
 Super Street Fighter II
 The Legend of Zelda: A Link to the Past (secret maze game)
```
"First of all, the Legend of Zelda wasn't the only cartridge that would activate the hidden maze game -- basically, any unsupported SNES cart would do it. I usually used Super Mario World."
Most of the above games, don't include any built-in Xband support, instead, Catapult reverse-engineered how they work, and patched them to work with the modem. Exceptions are Weaponlord and Doom, which were released with "modem support" (unknown what that means exactly... do they control modem I/O ports... interact with the modem BIOS... or are they patched the same way as other games, and the only difference is that the developers created the patches before releasing the game?)
Note: The japanese BIOS does read the Game cartridge header several times (unlike the US version which reads it only once), basically there is no good reason for those multiple reads, but it might indicate the japanese version includes multiple patches built-in in ROM?)

PCB "123-0002-16, Cyclone Rev 9, Catapult (C) 1995" Component List
```
  U1   28pin Winbond W24257S-70L                   (32Kx8 SRAM)
  U2   36pin X X, X BAND, X X, SNES XS? ROM 1.0.1  (BIOS ROM)
  U3  100pin ProdITH or FrodIIH?, H3A4D1049, 9511 Korea (with Hyundai logo)
  U4   68pin RC2324DPL, R6642, Rockwell 91, 9439 A49172-2, Mexico
  U5    6pin LITEON 4N25 (optocoupler) (near TN0) (back side)
  U6   28pin Winbond W24257S-70L                   (32Kx8 SRAM)
  U7    6pin AT&T LF1504 (solid state relay) (near TN0) (back side)
  BT0   2pin Battery (not installed) (component side)
  BT200 2pin Battery (3V Lithium Penata CR2430) (back side)
  SW1   3pin Two-position switch (purpose unknown... battery off ??)
  J0   10pin Card-reader (for credit cards or so?) 8 contacts, plus 2pin switch
  J1   62pin SNES Cartridge Edge (to be plugged into the SNES console)
  J2   62pin SNES Cartridge Slot (for game-cart plugged on top of the modem)
  J3  4/6pin RJ socket (to phone line)
  Y1    2pin Oscillator (24MHz or so?) (back side)
  TN0   4pin Transformator (671-8001 MIDCOM C439)
  LEDs       Three red LEDs (purpose/usage unknown?)
```

SNES Cart X-Band Smart Card Reader

The X-Band contains a built-in Smart Card reader (credit card shaped chip cards with 8 gold contacts). The X-Band BIOS contains messages that refer to "XBand Cards" and "XBand Rental Cards". There aren't any photos (or other info) of these cards in the internet, maybe X-Band requested customers to return the cards, or the cards got lost for another reason.

Purpose
Not much known. Reportedly the card reader was used for Prepaid Cards (for users whom didn't want Xband to charge the credit cards automatically), if that is correct, then only those users would have received cards, and other users didn't need to use the card reader? Note: The Options/Account Info screen show entries "Account" and "Card".

Smart Card I/O Ports
The BIOS seems to be accessing the cards via these I/O ports:
```
  FBC100h Card Data/Control/Whatever (out)
  FBC108h.Bit0 (In) Card Switch (1=card inserted, 0=card missing)
  FBC108h.Bit1 (In) Card Data (input)
```
Related BIOS functions are function 0380h..0386h.

Unknown 64bit Number
The Xband BIOS is reading a 64bit number via serial bus (which may or may not connect to the Smart Card) via two I/O Ports:
```
  FBC168h Data        ;bit2 (data, in/out)       ;\there is maybe also a reset
  FBC16Ah Direction   ;bit2 (0=input, 1=output)  ;/flag, eventually in bit5 ?
```
The sequence for reading the 64bits is somewhat like so:
```
  Data.Output(0), Delay (LOOPw0154h)
```

```
Data=Output(1), Delay (LOOPx01F4h)
Data=Input
wait until Data=1 or fail if timeout
wait until Data=0 or fail if timeout
wait until Data=1 or fail if timeout
Delay (LOOPx02BCh)
for i=1 to 4
  Data=Output(0), Delay (NOPx8)
  Data=Output(1), Delay (NOPx8)
  Data=Input, Delay (LOOPx0050h)
for i=1 to 4
  Data=Output(0), Delay (NOPx8)
  Data=Output(1), Delay (LOOPx003Ch)
  Data=Input, Delay (LOOPx001Eh)
Data=Input, Delay (LOOPx0064h)
for i=0 to 63
  Data=Output(1), Delay (NOPx8)
  Data=Input, Delay (LOOPx000Ah)
  key.bit(i)=Data, Delay (LOOPx004Bh)
```

For the exact timings (Delays and other software overload), see the BIOS function (at D7BE78h). Before doing the above stuff, the BIOS initializes [FBC168h]=40h, [FBC16Ah]=FFh (this may be also required).

The 64bit number is received LSB first, and stored in SRAM at 3FD8h-3FDFh. Whereas the last byte is a checksum across the first 7 bytes, calculated as so:

```
sum=00h
for i=0 to 55
  if (sum.bit(0) xor key.bit(i))=1 then sum=sum/2 xor 8Ch else sum=sum/2
```

For example, if the 7 bytes are "testkey", then the 8th byte must be 2Fh. Or, another simplier example would be setting all 8 bytes to 00h.

SNES Cart X-Band Rockwell Ports

Below are the I/O Ports of the Rockwell chip. In the SNES, Rockwell registers 00h-1Fh are mapped to EVEN memory addresses at FBC180h-FBC1BEh. The chip used in the SNES supports data/voice modem functions (but not fax modem functions).

FBC180h/FBC182h - 00h/01h - Receive Data Buffer

```
0-7  RBUFFER Received Data Buffer. Contains received byte of data
8    RXP     Received Parity bit (or ninth data bit)
9-15 N/A     Unused
```

FBC184h/FBC186h - 02h/03h - Control

```
0-8  N/A     Unused
9    GTE     TX 1800Hz Guard Tone Enable (CCITT configuration only)
10   SDIS    TX Scrambler Disable
11   ARC     Automatic on-line Rate Change sequence Enable
12   N/A     Unused
13   SPLIT   Extended Overspeed TX/RX Split. Limit TX to basic overspeed rate
14   HDLC    High Level HDLC Protocol Enable (in parallel data mode)
15   NRZIE   Unknown (listed in datasheet without further description)
```

FBC188h/FBC18Ah - 04h/05h - Control

```
0     CRFZ   Carrier Recovery Freeze. Disable update of receiver's carrier
             recovery phase lock loop
1     AGCFZ  AGC Freeze. Inhibit updating of receiver AGC
2     IFIX   Eye Fix. Force EYEX and EYEY serial data to be rotated
             equalizer output
3     EQFZ   Equalizer Freeze. Inhibit update of receiver's adaptive
             equalizer taps
4-5   N/A    Unused
6     SWRES  Software Reset. Reinitialize modem to its power turn-on state
7     EQRES  Equalizer Reset. Reset receiver adaptive equalizer taps to zero
8     N/A    Unused
9     TXVOC  Transmit Voice. Enable sending of voice samples
10    RCEQ   Receiver Compromise Equalizer Enable. Control insertion of
             receive passband digital compromise equalizer into receive path
11    CEQ(E) Compromise Equalizer Enable. Enable transmit passband digital
             compromise equalizer
12    TXSQ   Transmitter Squelch. Disable transmission of energy
13-15 N/A    Unused
```

FBC18Ch/FBC18Eh - 06h/07h - Control

```
0,1 WDSZ   Data Word Size, in asynchronous mode (5, 6, 7, or 8 bits)
2   STB    Stop Bit Number (number of stop bits in async mode)
3   PEN    Parity Enable (generate/check parity in async parallel data mode)
4,5 PARSL  Parity Select (stuff/space/even/odd in async parallel data mode)
6   EXOS   Extended Overspeed. Selects extended overspeed mode in async mode
7   BRKS   Break Sequence. Send of continuous space in parallel async mode
8   ABORT  HDLC Abort. Controls sending of continuous mark in HDLC mode
9   RA     Relay A Activate. Activate RADRV output
10  RB     Relay B Activate. Activate RBDVR output
11  L3ACT  Loop 3 (Local Analog Loopback) Activate. Select connection of
           transmitter's analog output Internally to receiver's analog input
12  N/A    Unused
13  L2ACT  Loop 2 (Local Digital Loopback) Activate. Select connection of
           receiver's digital output Internally to transmitter's digital
           input (locally activated digital loopback)
14  RDL    Remote Digital Loopback Request. Initiate a request for remote
           modem to go into digital loop-back
15  RDLE   Remote Digital Loopback Response Enable. Enable modem to respond
           to remote modem's digital loopback request
```

FBC190h/FBC192h - 08h/09h - Control

```
0   RTS    Request to Send. Request transmitter to send data
1   RTRN   Retrain. Send retrain-request or auto-rate-change to remote modem
2   N/A    Unused
3   TRFZ   Timing Recovery Freeze. Inhibit update of receiver's timing
           recovery algorithm
4   DDIS   Descrambler Disable. Disable receiver's descrambler circuit
5   N/A    Unused
6   TPPM   Transmitter Parallel Data Mode. Select parallel/serial TX mode
```

```
7   ASYNC  Asynchronous/Synchronous. Select sync/async data mode
8   SLEEP  Sleep Mode. Enter SLEEP mode (wakeup upon pulse on RESET pin)
9   N/A    Unused
10  DATA   Data Mode. Select idle or data mode
11  LL     Leased Line. Select leased line data mode or handshake mode
12  ORG    Originate. Select originate or answer mode (see TONEC)
13  DTMF   DTMF Dial Select. Select DTMF or Pulse dialing in dial mode
14  CC     Controlled Carrier. Select controlled or constant carrier mode
15  NV25   Disable V.25 Answer Sequence (Data Modes), Disable Echo Suppressor
           Tone (Fax Modes). Disable transmitting of 2100Hz CCITT answer tone
           when a handshake sequence is initiated in a data mode or disables
           sending of echo suppressor tone in a fax mode
```

FBC194h/FBC196h - 0Ah/0Bh - Status

```
0   CRCS   CRC Sending. Sending status of 2-byte CRC in HDLC mode
1-7 N/A    Unused
8   BEL103 Bell 103 Mark Frequency Detected. Status of 1270Hz Bell 103 mark
9   DTDET  DTMF Digit Detected. Valid DTFM digit has been detected
10  PNSUC  PN Success. Receiver has detected PN portion of training sequence
11  ATBELL Bell Answer Tone Detected. Detection status of 2225Hz answer tone
12  ATV25  V25 Answer Tone Detected. Detection status of 2100Hz answer tone
13  TONEC  Tone Filter C Energy Detected. Status of 1650Hz or 980Hz (selected
           by ORG bit) FSK tone energy detection by Tone C bandpass filter in
           Tone Detector configuration
14  TONEB  Tone Filter B Energy Detected. Status of 390Hz FSK tone energy
           detection by Tone B bandpass filter in Tone Detector configuration
15  TONEA  Tone Filter A Energy Detected. Status of energy above threshold
           detection by Call Progress Monitor filter in Dial Configuration or
           1300 Hz FSK tone energy detection by Tone A bandpass filter in
           Tone Detector configuration
```

FBC198h/FBC19Ah - 0Ch/0Dh - Status

```
0-3 DTDIG  Detected DTMF Digit. Hexadecimal code of detected DTMF digit
4-6 N/A    Unused
7   EDET   Early DTMF Detect. High group frequency of DTMF tone pair detected
8-9 N/A    Unused
10  SADET  Scrambled Alternating Ones Sequence Detected
11  U1DET  Unscrambled Ones Sequence Detected
12  SCR1   Scrambled Ones Sequence Detected
13  S1DET  S1 Sequence Detected
14  PNDET  Unknown (listed in datasheet without further description)
15  N/A    Unused
```

FBC19Ch/FBC19Eh - 0Eh/0Fh - Status

```
0-2 SPEED  Speed Indication. Data rate at completion of a connection
3   OE     Overrun Error. Overrun status of Receiver Data Buffer (RBUFFER)
4   FE     Framing Error. Framing error or detection of an ABORT sequence
5   PE     Parity Error. Parity error status or bad CRC
6   BRKD   Break Detected. Receipt status of continuous space
7   RTDET  Retrain Detected. Detection status of a retrain request sequence
8   FLAGS  Flag Sequence. Transmission status of Flag sequence in HDLC mode,
           or transmission of a constant mark in parallel asynchronous mode
9   SYNCD  Unknown (listed in datasheet without further description)
10  TM     Test Mode. Active status of selected test mode
11  RI     Ring Indicator. Detection status of a valid ringing signal
12  DSR    Data Set Ready. Data transfer state
13  CTS    Clear to Send. Training sequence has been completed (see TPDM)
14  FED    Fast Energy Detected. Energy above turn-on threshold is detected
15  RLSD   Received Line Signal Detector (carrier and receipt of valid data)
```

FBC1A0h/FBC1A2h - 10h/11h - Transmit Data Buffer

```
0-7   TBUFFER Transmitter Data Buffer. Byte to be sent in parallel mode
8     TXP     Transmit Parity Bit (or 9th Data Bit)
9-15  N/A     Unused
```

FBC1A4h/FBC1A6h - 12h/13h - Control

```
0-7   CONF   Modem Configuration Select. Modem operating mode (see below)
8-9   TXCLK  Transmit Clock Select (internal, disable, slave, or external)
10-11 VOL    Volume Control. Speaker volume (off, low, medium, high)
12-15 TLVL   Transmit Level Attenuation Select. Select transmitter analog
             output level attenuation In 1 dB steps. The host can fine tune
             transmit level to a value lying within a 1 dB step In DSP RAM
```

FBC1A8h/FBC1AAh - 14h/15h - Unused

```
0-15  N/A    Unused
```

FBC1ACh/FBC1AEh - 16h/17h - Y-RAM Data (16bit)
FBC1B0h/FBC1B2h - 18h/19h - X-RAM Data (16bit)

```
0-15  DATA  RAM data word (R/W)
```

FBC1B4h/FBC1B6h - 1Ah/1Bh - Y-RAM Addresss/Control
FBC1B8h/FBC1BAh - 1Ch/1Dh - X-RAM Addresss/Control

```
0-8   ADDR  RAM Address
9     WT    RAM Write (controls read/write direction for RAM Data registers)
10    CRD   RAM Continuous Read. Enables read of RAM every sample from
            location addressed by ADDR Independent of ACC and WT bits
11    IOX   X-RAM only: I/O Register Select. Specifies that X RAM ADDRESS
            bit0-7 (Port 1Ch) is an internal I/O register address
11    N/A   Y-RAM only: Unused
12-14 N/A   Unused
15    ACC   RAM Access Enable. Controls DSP access of RAM associated with
            address ADDR bits. WT determines if a read or write is performed
```

FBC1BCh/FBC1BEh - 1Eh/1Fh - Interrupt Handling

```
0   RDBF    Receiver Data Buffer Full (RBUFFER Full)
1   N/A     Unused
2   RDBIE   Receiver Data Buffer Full Interrupt Enable
3   TDBE    Transmitter Data Buffer Empty (TBUFFER Empty)
```

```
4   N/A     Unused
5   TDBIE   Transmitter Data Buffer Empty Interrupt
6   RDBIA   Receiver Data Buffer Full Interrupt Active (IRQ Flag)
7   TDBIA   Transmitter Data Buffer Empty Interrupt Active (IRQ Flag)
8   NEWC    New Configuration. Initiates new configuration (cleared by modem
9   N/A     Unused                   upon completion of configuration change)
10  NCIE    New Configuration Interrupt Enable
11  NEWS    New Status. Detection of a change in selected status bits
12  NSIE    New Status Interrupt Enable
13  N/A     Unused
14  NCIA    New Configuration Interrupt Active (IRQ Flag)
15  NSIA    New Status Interrupt Active (IRQ Flag)
```

CONF Values

Rockwell didn't describe the CONF values in their RC96V24DP/RC2324DPL datasheet, below CONF values are taken from a RC96DT/RC144DT datasheet (hoping that the values are same, though the higher baudrates obviously won't work).

```
CONF  Bits/sec Mode Name
01h   2400     V.27 ter
02h   4800     V.27 ter
11h   4800     V.29
12h   7200     V.29
14h   9600     V.29
52h   1200     V.22
51h   600      V.22
60h   0-300    Bell 103
62h   1200     Bell 212A
70h   -        V.32 bis/V.23 clear down    ;\
71h   4800     V.32                        ;
72h   12000    V.32 bis TCM                ; RC96DT/RC144DT only
74h   9600     V.32 TCM                    ; (not RC96V24DP/RC2324DPL)
75h   9600     V.32                        ;
76h   14400    V.32 bis TCM                ;
78h   7200     V.32 bis TCM                ;/
80h   -        Transmit Single Tone
81h   -        Dialing                  ;used by SNES X-Band (dial mode)
82h   1200     V.22 bis
83h   -        Transmit Dual Tone
84h   2400     V.22 bis                 ;used by SNES X-Band (normal mode)
86h   -        DTMF Receiver
A0h   0-300    V.21
A1h   75/1200  V.23 (TX/RX)
A4h   1200/75  V.23 (TX/RX)
A8h   300      V.21 channel 2
B1h   14400    V.17 TCM                    ;\
B2h   12000    V.17 TCM                    ; RC96DT/RC144DT only
B4h   9600     V.17 TCM                    ; (not RC96V24DP/RC2324DPL)
B8h   7200     V.17 TCM                    ;/
```

XBand X/Y RAM Rockwell

Below are X/Y RAM addresses that can be accessed via Ports 16h-1Dh.
Addresses 000h-0FFh are "Data RAM", 100h-1FFh are "Coefficient RAM".
X-RAM is "Real RAM", Y-RAM is "Imaginary RAM" (whatever that means).

```
XRAM     YRAM     Parameter
032      -        Turn-on Threshold
03C      -        Lower Part of Phase Error (this, in X RAM ?)
-        03C      Upper Part of Phase Error (this, in Y RAM ?)
-        03D      Rotation Angle for Carrier Recovery
03F      -        Max AGC Gain Word
049      049      Rotated Error, Real/Imaginary
059      059      Rotated Equalizer Output, Real/Imaginary
05E      05E      Real/Imaginary Part of Error
06C      -        Tone 1 Angle Increment Per Sample (TXDPHI1)
06D      -        Tone 2 Angle Increment Per Sample (TXDPHI2)
06E      -        Tone 1 Amplitude (TXAMP1)
06F      -        Tone 2 Amplitude (TXAMP2)
070      -        Transmit Level Output Attenuation
071      -        Pulse Dial Interdigit Time
072      -        Pulse Dial Relay Make Time
073      -        Max Samples Per Ring Frequency Period (RDMAXP)
074      -        Min Samples Per Ring Frequency Period (RDMINP)
07C      -        Tone Dial Interdigit Time
07D      -        Pulse Dial Relay Break Time
07E      -        DTMF Duration
110-11E  100-11E  Adaptive Equalizer Coefficients, Real/Imag.
110      100      First coefficient, Real/Imag. (1) (Data/Fax)
110      110      Last Coefficient, Real/Imag. (17) (Data)
11E      11E      Last Coefficient, Real/Imag. (31) (Fax)
-        121      RLSD Turn-off Time
12D      -        Phase Error
12E      -        Average Power
12F      -        Tone Power (TONEA)
130      -        Tone Power (TONEB,ATBELL,BEL103)
131      -        Tone Power (TONEC,ATV25)
136      -        Tone Detect Threshold for TONEA               (THDA)
137      -        Tone Detect Threshold for TONEB,ATBELL,BEL103 (THDB)
138      -        Tone Detect Threshold for TONEC,ATV25         (THDC)
13E      -        Lower Part of AGC Gain Word
13F      -        Upper Part of AGC Gain Word
152      -        Eye Quality Monitor (EQM)
-        162-166  Biquad 5 Coefficients a0,a1,a2,b1,b2
-        167-16B  Biquad 6 Coefficients a0,a1,a2,b1,b2
-        16C-170  Biquad 1 Coefficients a0,a1,a2,b1,b2
-        171-175  Biquad 2 Coefficients a0,a1,a2,b1,b2
-        176-17A  Biquad 3 Coefficients a0,a1,a2,b1,b2
179      -        Turn-off Threshold
-        17B-17F  Biquad 4 Coefficients a0,a1,a2,b1,b2
```

SNES Cart X-Band Rockwell Notes

Configuration Changes
Various changes (to ASYNC, WDSZ, etc.) seem to be not immediately applied. Instead, one must apply them by setting NEWC=1 by software (and then wait until hardware sets NEWC=0).

Dialing
Dialing is done by setting CONF=81h, and then writing the telephone number digits (range 00h..09h) to TBUFFER; before each digit wait for TDBE=1 (TX buffer empty), the BIOS also checks for TONEA=1 before dialing.
The telephone number for the X-Band server is stored as ASCII string in the BIOS ROM:

```
"18002071194"  at D819A0h in US-BIOS (leading "800" = Toll-free?)
"03-55703001"  at CE0AB2h in Japanese BIOS (leading "3" = Tokyo?)
```

Notes: Before dialing the above 'ASCII' numbers, the US-BIOS first dials 0Ah,07h,00h, and the japanese one first dials 01h. The "-" dash in the japanese string isn't dialed.

Offline
There seems to be no explicit offline mode (in CONF register). Instead, one must probably change the Relay A/B bits (RA/RB) to go online/offline.

XBand Pin-Outs Rockwell

```
Pin Number Signal Name I/O Type
1 RS2 IA
2 RS1 IA
3 RS0 IA
4 /TEST1
5 /SLEEP OA
6 RING
7 EYEY OB
8 EYEX OB
9 EYESYNC OB
10 RESET ID
11 XTLI IE
12 XTLO OB
13 +5VD
14 GP18 OA
15 GP16 OA
16 XTCLK IA
17 DGND1
18 TXD IA
19 TDCLK OA
20 TRSTO MI
21 TSTBO MI
22 TDACO MI
23 RADCI MI
24 RAGCO MI
25 MODEO MI
26 RSTBO MI
27 RRSTO MI
28 /RDCLK OA
29 RXD OA
30 TXA2 O(DD)
31 TXA1 O(DD)
32 RXA I(DA)
33 RFILO MI
34 AGCIN MI
35 VC
36 NC
37 NC
38 NC
39 /RBDVR OD
40 AGND
41 /RADRV OD
42 /SLEEP1 IA
43 RAGCI MI
44 NC
45 RSTBI MI
46 RRSTI MI
47 RADCO MI
48 TDACI MI
49 TRSTI MI
50 TSTBI MI
51 MODE1 MI
52 +5VA
53 SPKR O(OF)
54 DGND2
55 D7 IA/OB
56 D6 IA/OB
57 D5 IA/OB
58 D4 IA/OB
59 D3 IA/OB
60 D2 IA/OB
61 D1 IA/OB
62 D0 IA/OB
63 /IRQ OC
64 /WRITE IA
65 /CS IA
66 /READ IA
67 RS4 IA
68 RS3 IA
```

Notes:
(1) MI = Modem Interconnection
(2) NC = No connection (may have internal connection; leave pin disconnected (open).
(3) I/O types are described in Table 2-3 (digital signals) and Table 2-4 (analog signals).

SNES Cart FLASH Backup

Most SNES games are using battery-backed SRAM for storing data, the only exception - which do use FLASH memory - are the JRA PAT BIOS cartridges for the SFC Modem.

SNES Add-On SFC Modem (for JRA PAT)

There are two JRA PAT versions, the older one (1997) supports only AMD FLASH, the newer one (1999) supports AMD/Atmel/Sharp FLASH chips.

```
ID=2001h - AM29F010 AMD (128Kbyte)      ;supported by BOTH bios versions
ID=D51Fh - AT29C010A Atmel (128Kbyte)   ;supported only by newer bios version
ID=32B0h - LH28F020SUT Sharp (256Kbyte?);supported only by newer bios version
```

The FLASH Size size defined in entry [FFBCh] of the Cartridge Header (this is set to 07h in JRA PAT, ie. "(1K SHL 7)=128Kbytes").
There don't seem to be any data sheets for the Sharp LH28F020SUT-N80 chip (ID B0h,32h) (so not 100% sure if it's really 256Kbytes), anyways, it does somehow resemble LH28F020SU-N (5V, ID B0h,30h) and LH28F020SU-L (5V/3.3V, ID B0h,31h).

JRA PAT Memory Map

```
80h-9Fh:8000h-FFFFh  ;1Mbyte LoROM (broken into 32 chunks of 32Kbytes)
C0h-C3h:0000h-7FFFh  ;128Kbyte FLASH (broken into 4 chunks of 32Kbytes)
```

AMD FLASH

Get Device ID (Type 1 - AMD)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=90h           ;enter ID mode
manufacturer=01h=[C00000h], device_type=20h=[C00001h] ;read ID (AM29F010)
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h           ;terminate command
```

Erase Entire Chip (Type 1 - AMD)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=80h           ;prepare erase
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=10h           ;erase entire chip
repeat, stat=[C00000h], until stat.bit7=1=okay, or stat.bit5=1=timeout
```

Erase 16Kbyte Sector (Type 1 - AMD)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=80h           ;prepare erase
[C05555h]=AAh, [C02AAAh]=55h, [Cxx000h]=30h           ;erase 16kbyte sector
repeat, stat=[Cxx000h], until stat.bit7=1=okay, or stat.bit5=1=timeout
```

Write Single Data Byte (Type 1 - AMD)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=A0h           ;write 1 byte command
[Cxxxxxh]=dta                                         ;write the data byte
repeat, stat=[Cxxxxxh], until stat.bit7=dta.bit7=okay, or stat.bit5=1=timeout
```

Notes

After AMD timeout errors, one should issue one dummy/status read from [C00000h] to switch the device back into normal data mode (at least, JRA PAT is doing it like so, not too sure if that is really required/correct).

ATMEL FLASH

Get Device ID (Type 2 - Atmel)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=90h           ;enter ID mode
manufacturer=1Fh=[C00000h], device_type=D5h=[C00001h] ;read ID (AT29C010A)
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=F0h           ;terminate command
```

Erase Entire Chip (Type 2 - Atmel)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=80h           ;prepare erase
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=10h           ;erase entire chip
wait two frames (or check if bit6 toggles on each read from [C00000h])
```

Erase 16Kbyte Sector (Type 2 - Atmel)

```
No such command (one can write data without erasing)
(to simulate a 16K-erase: write 128 all FFh-filled 128-byte blocks)
```

Write 1..128 Data Bytes (within 128-byte boundary) (Type 2 - Atmel)

```
[C05555h]=AAh, [C02AAAh]=55h, [C05555h]=A0h           ;write 1..128 byte(s)
[Cxxxxxh+0..n]=dta[0..n]                              ;write the data byte(s)
repeat, stat=[Cxxxxxh+n], until stat=dta[n]           ;wait last written byte
```

Notes

The JRA PAD functions do include a number of wait-vblank delays between various ATMEL commands, that delays aren't shown in above flowcharts.

SHARP FLASH

Get Device ID (Type 3 - Sharp)

```
[C00000h]=90h                                         ;enter ID mode
manufacturer=B0h=[C00000h], device_type=32h=[C00001h] ;read ID (LH28F020SUT)
[C00000h]=FFh                                         ;terminate command
```

Set/Reset Protection (Type 3 - Sharp)

```
[C00000h]=57h/47h, [C000FFh]=D0h    ;<-- C000FFh (!)  ;set/reset protection
repeat, stat=[C00000h], until stat.bit7=1             ;wait busy
if stat.bit4=1 or stat.bit5=1 then [C00000h]=50h      ;error --> clear status
[C00000h]=FFh                                         ;terminate command
```

Erase Entire Chip (Type 3 - Sharp)

```
[C00000h]=A7h, [C00000h]=D0h                          ;erase entire chip
repeat, stat=[C00000h], until stat.bit7=1             ;wait busy
if stat.bit4=1 or stat.bit5=1 then [C00000h]=50h      ;error --> clear status
[C00000h]=FFh                                         ;terminate command
```

Erase 16Kbyte Sector (Type 3 - Sharp)

```
[C00000h]=20h, [Cxx000h]=D0h                          ;erase 16kbyte sector
repeat, stat=[C00000h], until stat.bit7=1             ;wait busy
if stat.bit4=1 or stat.bit5=1 then [C00000h]=50h      ;error --> clear status
[C00000h]=FFh                                         ;terminate command
if failed, issue "Reset Protection", and retry
```

Write Single Data Byte (Type 3 - Sharp)

```
[C00000h]=40h                                         ;write 1 byte command
[Cxxxxxh]=dta                                         ;write the data byte
repeat, stat=[C00000h], until stat.bit7=1             ;wait busy
```

```
;below error-check & terminate are needed only after writing LAST byte
if stat.bit4=1 or stat.bit5=1 then [C00000h]=50h      ;error --> clear status
[C00000h]=FFh                                        ;terminate command
```

PCB VERSIONS

Older PCB "SHVC-1A9F-01" (1996) (DIP) (for JRA-PAT and SPAT4)

```
U1 32pin ROM
U2 32pin AMD AM29F010-90PC (FLASH)
U3 16pin SN74LS139AN
U4 16pin D411B (CIC)
```

Newer PCB "SHVC-1A8F-01" (1999) (SMD) (for JRA-PAT-Wide)

```
U1 32pin ROM
U2 32pin Sharp LH28F020SUT-N80 (FLASH)
U3 16pin 74AC139
U4 18pin F411B (CIC)
U5 14pin 74AC08
```

See Also

Another approach for using FLASH backup is used in carts with Data Pack slots:
SNES Cart Data Pack Slots (satellaview-like mini-cartridge slot)

SNES Cart Cheat Devices

Code Format Summary

```
Pro Action Replay         AAAAAADD        raw 8-digits         WRAM
Pro Action Replay Mk2/Mk3 AAAAAADD        raw 8-digits         WRAM/ROM/SRAM
X-Terminator/Game Wizard  AAAAAADD        raw 8-digits         WRAM
Game Genie/Game Mage      DDAA-AAAA       encrypted 4-4 digits ROM/SRAM
Gold Finger               AAAAADDDDDDCCW  raw 14-digits        DRAM/SRAM
Front Far East            NNAAAAAADD..    raw 10..80 digits    DRAM offset
```

Code Format Details

SNES Cart Cheat Devices - Code Formats

Hardware Details

SNES Cart Cheat Devices - Game Genie
SNES Cart Cheat Devices - Pro Action Replay I/O Ports
SNES Cart Cheat Devices - Pro Action Replay Memory
SNES Cart Cheat Devices - X-Terminator & Game Wizard
SNES Cart Cheat Devices - Game Saver
SNES Cart Cheat Devices - Theory

Cheat Devices & Number of Hardware/Software patches & Built-in codes

```
Name                           Hardware/ROM  Software/WRAM  Built-in
Pro Action Replay              None  (of 4)  4              None
Pro Action Replay Mk2a/b       0/2/4 (of 4)  100            None
Pro Action Replay Mk3          1/5   (of 7)  100            ? games
Game Genie (Codemasters/Galoob) 5    (of 6)  None           None
Game Mage (Top Game & Company) 8?            None?          250 codes?
X-Terminator (Fire)            None  (of 0)  4              None
X-Terminator 2 (noname)        None  (of 0)  64             307 games
Game Wizard (Innovation)       None?         ?              ?
Game Saver (Nakitek) allows to save WRAM/VRAM snapshots in non-battery DRAM
Game Saver+ (Nakitek) allows to save WRAM/VRAM snapshots in battery DRAM
Super UFO (copier, supports Gold Finger and X-Terminator codes)
Super Wild Card/Magicom (copiers, support Gold Finger and Front Far East)
Parame ROM Cassette Vol 1-5 (by Game Tech) (expansions for X-Terminator 2)
```

Note: The Game Mage's stylished "GAME|~AGE)" logo is often misread as "Gametaged".

Links

http://www.gamegenie.com/cheats/gamegenie/snes/index.html
http://www.world-of-nintendo.com/pro_action_replay/super_nes.shtml
http://www.gamefaqs.com/snes/562623-harvest-moon/faqs/10690
http://www.gamefaqs.com/snes/588741-super-metroid/faqs/5667

SNES Cart Cheat Devices - Code Formats

PAR AAAAAADD - Normal Pro Action Replay Codes (Datel)

The Pro Action Replay is a cheat device for the SNES produced by Datel. The original PAR only support 3 codes, but the PAR2 supports 255 and has a built-in trainer for code searcher. There is also a PAR3, but the added features are unknown.

```
AAAAAADD  ;-address (AAAAAA) and data (DD)
```

Address can be a ROM, SRAM, or WRAM location. Patching cartridge memory (both ROM and SRAM) is implemented by hardware (supported by PAR2-PAR3 only, not by PAR1 or X-Terminator 1-2). Patching WRAM is done by software (rewriting the values on each Vblank NMI). WRAM addresses must be specified as 7E0000h-7FFFFFh (mirrors at nn0000h-nn1FFFh aren't recognized by the BIOSes).

PAR 7E000000 - Do nothing

This is the most important PAR code (required as padding value, since the GUI doesn't allow to remove items from the code list):

PAR FE0000xx..FFFFFFxx - Pre-boot WRAM patch (PAR1 only)

Writes xx to the corresponding WRAM address at 7E0000h..7FFFFFh, this is done only once, and it's done BEFORE starting the game (purpose unknown - if any).

PAR 00600000 - Disable Game's NMI handler (PAR3 only)

Disables the game's NMI handler (executes only the NMI handler of the PAR BIOS).

PAR DEADC0DE - Special Multi-Byte Code Prefix (PAR2a/b and PAR3 only)

Allows to hook program code, this feature is rarely used.

```
DEADC0DE  ;-prefix (often misspelled as "DEADCODE", with "O" instead "0")
AAAAAANN  ;-address (AAAAAA) and number of following 4-byte groups (NN)
DDEEFFGG  ;-first 4-byte group    (DD=1st byte, .. GG=4th byte)
HHIIJJKK  ;-second 4-byte group   (HH=5th byte, .. KK=8th byte) (if any)
...       ;-further 4-byte groups (etc.)                        (if any)
```

The data portion (DD,EE,FF..) (max 62h*4 = 188h bytes) is relocated to SRAM (in the PAR cartridge), and the ROM address AAAAAA is patched by a 4-byte "JMP nnnnnn" opcode (doing a far jump to the address of the relocated SRAM code; this would be at 006A80h in PAR3, at 006700h in PAR2a/b, and isn't supported in PAR1). There seems to be no special action required when returning control to the game (such like disabling the SRAM - if the hardware does support that at all?) (or such actions are required only for HiROM games that have their own SRAM at 6000h?) (or games are typically accessing SRAM at 306xxxh, so there is no conflict with PAR memory at 006xxxh?).
One can use only one DEADC0DE at a time, and, when using it, there are some more restrictions: On PAR2a/b one cannot use ANY other hardware/software patches. On PAR3 one can keep using ONE hardware patch (and any number of software patches).

PAR C0DEnn00 - Whatever (X-Terminator 2 only - not an official PAR code)
Somehow changes the NMI (and IRQ) handling of the X-Terminator 2, "nn" can be 00..06.

Game Genie Codes (Codemasters/Galoob)

```
DDAA-AAAA  ;-encrypted data (DD) and encrypted/shuffled address (AA-AAAA)
```

Address can be a ROM, or SRAM location (internal WRAM isn't supported). To decrypt the code, first replace the Genie Hex digits by normal Hex Digits:

```
Genie Hex:    D  F  4  7  0  9  1  5  6  B  C  8  A  2  3  E
Normal Hex:   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

Thereafter, DD is okay, but AAAAAA still needs to be deshuffled:

```
ijklqrst opabcduv wxefghmn   ;Genie Address (i=Bit23 ... n=Bit0)
abcdefgh ijklmnop qrstuvwx   ;SNES Address (a=Bit23 ... x=Bit0)
```

Aside from being generally annoying, the encryption makes it impossible to make codes like "Start in Level NN" (instead, one would need to make separate codes for each level).
Game Genie codes can be reportedly also used with the Game Mage. And, the PAR3 includes a "CONV" button for converting Game Genie codes. When manually decrypting them, Game Genie codes would also work on PAR2 (although the PAR2 won't allow to use five ROM patches at once).

Gold Finger / Goldfinger Codes (unknown who created this format)
These codes are rarely used, and there isn't much known about them. Reportedly, they have been supported by "certain copiers" (unknown which ones... Super UFO, and also copiers from Front Far East?).

```
AAAAADDEEFFCCW  ;-Address (AAAAA), Data (DD,EE,FF), Checksum (CC), Area (W)
```

The Address is a ROM address, not a CPU address. Data can be 1-3 bytes, when using less than 3 bytes, pad the EE,FF fields with "XX" (and treat them as zero in the checksum calculation). Checksum is calculated as so:

```
CC = A0h + AAAAA/10000h + AAAAA/100h + AAAAA + DD (+ EE (+ FF))
```

W tells the copier whether to replace the byte in the DRAM (ROM image) or the SRAM (Saved game static RAM) of the copier:

```
W=0  DRAM (ROM image) (reportedly also for W=2,8,A,C,F)
W=1  SRAM (Saved game image)
```

The 5-digit address allows to access max 1Mbyte. The address is an offset within the ROM-image (excluding any 200h-byte header). The first LoROM byte would be at address 00000h. Some copiers are using interleaved HiROM images - unknown if any such interleave is used on code addresses - if so, first HiROM byte would be at "ROMsize/2" (in middle of ROM-image), otherwise it'd be at 00000h (at begin at ROM-image).
Note: It doesn't seem to be possible to enter "X" digits in all copiers. Double Pro Fighter Q allows to enter "X".

Front Far East Codes (Front Far East)
Supported by Front Far East copiers (Super Magicom and/or Super Wild Card?). This format is even less popular than the Gold Finger format.

```
NNAAAAAADD..    Number of bytes (NN), Address (AAAAAA), Data (DD..)
```

Allows to change 1..36 bytes; resulting in a code length of 10..80 digits (to avoid confusion with 14-digit Gold Finger codes, Front Far East wants Gold Finger codes to be prefixed by a "G").
AAAAAA is a 24bit offset within the ROM-image (excluding any 200h-byte header). As far as known, Front Far East didn't use interleaved ROM-images, so the offset should be straight.

SNES Cart Cheat Devices - Game Genie

Game Genie BIOS Versions
There are at least three BIOS versions, named "GENSRC", "K7", and "ed":

```
"GENSRC"  32Kbytes LoROM, straight I/O addresses  CRC32=AC94F94Ah
"K7"      32Kbytes LoROM, messy I/O addresses     CRC32=F8D4C303h
"ed"      64Kbytes LoROM, messy I/O addresses     CRC32=58CBF2FEh
```

The names are stored at ROM-offset 7FE0h (aka SNES address 00FFE0h). Most of the cartridge header contains garbage (no checksum, wrong ROM size, etc.), only the 21-byte title field is (more or less) correct:

```
"Game Genie      ",0,0,0,0,0   ;32K versions (GENSRC & K7)
"Game Genie         Jo"        ;64K version (ed)
```

Note: All three versions support only 5 codes to be entered.

Game Genie I/O Ports
The "GENSRC" version uses quite straight I/O addresses, the "K7" and "ed" versions have the addresses messed up (unused gaps between the codes, several mirrors, of which, mirrors with address bit8 and bits16-23 all zero are having address bit0 inverted).

```
Version   "GENSRC"          "K7" & "ed"
Control   W:008000h         W:xx8100h, W:008001h  ;ed: xx=00, K7: xx=FF
CodeFlags R/W:008001h       R:FF8001h, W:008000h  ;bit 0-4 = enable code 1-5
CodeMsb   R/W:008003h+N*4   R:FF8005h+N*6, W:008004h+N*6  ;\
CodeMid   R/W:008004h+N*4   R:FF8006h+N*6, W:008007h+N*6  ; N=0-4 for
CodeLsb   R/W:008005h+N*4   R:FF8007h+N*6, W:008006h+N*6  ; code 1-5
CodeData  R/W:008006h+N*4   R:FF8008h+N*6, W:008009h+N*6  ;/
```

Other (accidently) used I/O addresses are: W:004017h (bugged joypad access), and W:00FFEAh/00FFEBh (used in an attempt to install a bugged NMI handler with RET instead of RETI opcode; the hardware is hopefully ignoring that attempt).

Control Register
Allows to select what is mapped to memory. Used values are:

```
00h Select Game Genie BIOS
02h Select Game Genie I/O Ports
06h Select Game Cartridge
07h Select Game Cartridge and keep it selected
```

This register is set to 00h on /RESET (warmboot & coldboot).

Code Flags Register
Used to enable the 5 codes:

```
  Bit0-4  Enable Code 1..5  (0=Disable, 1=Enable)
  Bit5-7  Should be zero
```
This register is set to 00h on initial power-up (coldboot), but kept intact on /RESET (warmboot), allowing to restore the previously enabled codes.

CodeAddress/CodeData Registers (5*4 bytes)
Contains the 24bit address and 8bit data values (in decrypted form). The registers are kept intact on /RESET (warmboot), allowing to restore the previously entered codes - however, when restoring the codes, the "K7" and "ed" BIOS versions are ORing the LSB of of the 24bit address value with 01h, thereby destroying all codes with even addresses (unknown why that's been done).

Chipset
The exact chipset is unknown, there should be the ROM and some logic (and no SRAM). There is also a LED and a 2-position (?) switch (unknown function).

XXX
For more in-depth info see "genie.txt" from Charles MacDonald.
```
  http://cgfm2.emuviews.com/txt/genie.txt
```

Game Genie 2 (prototype)
There is also an unreleased Game Genie 2 prototype, the thing includes a small LCD screen, five push buttons, four LEDs, battery backed SRAM, a 8255 PIO, a huge ACTEL chip, and some expansion connectors.

SNES Cart Cheat Devices - Pro Action Replay I/O Ports

PAR1 I/O Ports (W)
```
  008000h                 ;Code 0-3 (MID) (shared for code 0..3)
  010000h,010001h,010002h  ;Code 0 (DTA,LSB,MSB) (not used by BIOS)
  010003h                 ;Control (set to FFh)
  010004h,010005h,010006h  ;Code 1 (DTA,LSB,MSB) (not used by BIOS)
  010007h,010008h,010009h  ;Code 2 (DTA,LSB,MSB) (used as NMI vector.LSB)   01000Ah,01000Bh,01000Ch  ;Code 3 (DTA,LSB,MSB) (used as NMI vector.MSB)
```
Most I/O ports are overlapping WRAM in bank 01h (unlike PAR2-PAR3 which use bank 10h). The four code registers would allow to apply 4 hardware patches, the PAR1 BIOS actually has provisions for doing that, but, before applying those patches it erases code 0-3 (and does then use code 2-3 for patching the NMI vector at 00FFEAh for applying the codes as WRAM software patches).
The PAR1 supports only LoROM addresses (address bit15 removed, and bit23-16 shifted down). The PAR1 does not (maybe cannot) disable unused codes; instead, it directs them to an usually unused ROM location at 00FFF6h (aka 007FF6h after removing bit15). Applying the codes is done in order DTA,MID,LSB,MSB, whereof, the "shared" MID value is probably applied on the following LSB port write.
The control register is set to FFh before starting the game, purpose is unknown (maybe enable the codes, or write-protect them, or disable the BIOS ROM; in case that can be done by software).

PAR2 I/O Ports (W)
```
  100000h,100001h,100002h,100003h  ;Code 0 (DTA,LSB,MID,MSB) (code 0)
  100004h,100005h,100006h,100007h  ;Code 1 (DTA,LSB,MID,MSB) (code 1)
  100008h,100009h,10000Ah,10000Bh  ;Code 2 (DTA,LSB,MID,MSB) (code 2/NMI.LSB)
  10000Ch,10000Dh,10000Eh,10000Fh  ;Code 3 (DTA,LSB,MID,MSB) (code 3/NMI.MSB)
  100010h      ;Control A (set to 00h or FFh)
  C0A00nh      ;Control B (address LSBs n=0..7) (written data=don't care)
```
The registers overlapping the WRAM area are similar as for PAR1. The PAR2 BIOS allows to use the code registers for ROM patches (and/or hooking the NMI handler for WRAM patches).
Similar as in PAR1, address bit23-16 are shifted down, but with bit15 being moved to bit23 (still a bit messy, but HiROM is now supported). Unused codes are redirected to 00FFF6h (aka 807FF6h after moving bit15).
The control register is set to FFh before starting the game, purpose is unknown (maybe enable the codes, or write-protect them, or disable the BIOS ROM; in case that can be done by software).
The lower address bits of the newly added C0A00nh Register are:
```
  Address bit0 - set if one or more codes use bank 7Fh..FEh
  Address bit1 - set/cleared for PAL/NTSC selection (or vice-versa NTSC/PAL?)
  Address bit2 - set to... maybe, forcing the selection in bit1 (?)
```
The exact purpose of that three bits is unknown (and their implemention in PAR2a/b BIOSes looks bugged). Doing anything special on bank 7Fh-FEh doesn't make any sense, maybe the programmer wanted to use banks 80h-FFh, but that wouldn't make much more sense either; it might be something for enabling/disabling memory mirrors or so. The default PAL/NTSC flag is auto-detected by reading 213Fh.Bit4 (the BIOS is doing that detection twice, with opposite results on each detection, which seems to be a bug), the flag can be also manually changed in the BIOS menu; purpose of the PAL/NTSC thing is unknown... maybe directing a transistor to shortcut D4 to GND/VCC when games are reading 213Fh.

PAR3 I/O Ports
```
  100000h,100001h,100002h,100003h  ;Code 0 (DTA,LSB,MID,MSB) (code 0)
  100004h,100005h,100006h,100007h  ;Code 1 (DTA,LSB,MID,MSB) (code 1)
  100008h,100009h,10000Ah,10000Bh  ;Code 2 (DTA,LSB,MID,MSB) (code 2)
  10000Ch,10000Dh,10000Eh,10000Fh  ;Code 3 (DTA,LSB,MID,MSB) (code 3)
  100010h,100011h,100012h,100013h  ;Code 4 (DTA,LSB,MID,MSB) (code 4)
  100014h,100015h,100016h,100017h  ;Code 5 (DTA,LSB,MID,MSB) (always NMI.LSB)
  100018h,100019h,10001Ah,10001Bh  ;Code 6 (DTA,LSB,MID,MSB) (always NMI.MSB)
  10001Ch         ;Control A    (bit4,6,7)
  10001Dh-10001Fh ;Set to zero  (maybe accidently, trying to init "code 7")
  10003Ch         ;Control B    (set to 01h upon game start)
  086000h         ;Control LEDs (bit0,1)
  206000h         ;Control C    (bit0)
  008000h         ;Control D    (set to 00h upon PAR-NMI entry)
```
Control A (10001Ch):
```
  Bit0-3 Should be 0
  Bit4   ROM Mapping (0=Normal, 1=Temporarily disable BIOS & enable GAME ROM)
  Bit5   Should be 0
  Bit6-7 Select/force Video Type (0=Normal, 1=NTSC, 2=PAL, 3=Reserved)
```
Control LEDs (086000h) (LEDs are in sticker area on front of cartridge):
```
  Bit0   Control left or right LED? (0=on or off?, 1=off or on?)
  Bit1   Control other LED          ("")
  Bit2-7 Should be 0
```
Control C (206000h):
```
  Bit0   Whatever (0=BIOS or PAR-NMI Execution, 1=GAME Execution)
  Bit1-7 Should be 0
```
Code0-6:
Unused codes are set to 00000000h (unlike PAR1/PAR2), and, codes use linear 24bit addresses (without moving/removing bit15). Of the seven codes, code 5-6 are always used for hooking the NMI handler (even when not using any WRAM software patches), so one can use max 5 hardware ROM patches.

SNES Cart Cheat Devices - Pro Action Replay Memory

PAR1-PAR3 SRAM
All PAR versions contain 32Kbytes SRAM, divided into 8K chunks, which are unconventionally mapped to EVEN bank numbers.
```
00/02/04/06:6000h..7FFFh      ;-32Kbyte SRAM (four 8K banks)
```
The SRAM is used as internal workspace (stack & variables, code list, NMI handler, deadcode handler, and list of possible-matches for the code finder).
Unknown if the SRAM is battery backed (the way how it is used by the BIOS suggests that it is NOT battery backed).
Note: Many HiROM games have their own SRAM mapped to 6000h-7FFFh, unknown if/how/when the PAR can disable its SRAM for compatibility with such games (PAR1 seems to be designed for LoROM games only, but newer PAR2-PAR3 <should> have HiROM support - if so, then the hardware must somehow switch the SRAM on/off depending on whether it executes game code, or PAR code like NMI & deadcode handlers).

PAR1-PAR3 Switch
All PAR versions do have a 3-position switch (on the right edge of the cartridge). The way how Datel wants it to be used seems to be:
```
1) Boot game with switch in MIDDLE position (maybe needed only for testing)
2) Set LOWER position & push RESET button (to enter the BIOS menu)
3) After selecting codes/cheat finder, start game with MIDDLE position
4) Finally, UPPER position enables codes (best in-game, AFTER intro/menu)
```
Technically, the switch seems to work like so:
```
UPPER Position   "Codes on"    Enable GAME and enable codes
MIDDLE Position  "Codes off"   Enable GAME and disable codes
LOWER Position   "Trainer On"  Enable BIOS and (maybe) enable codes
```
The "Codes off" setting may be required for booting some games (which may use WRAM for different purposes during intro & game phases). The purpose of the "Trainer" setting is unclear, GAME/BIOS mapping could be as well done via I/O ports (and at least PAR3 does actually have such a feature for reading the GAME header during BIOS execution).
There seems to be no I/O ports for sensing the switch setting, however, the "Codes off" setting can be sensed by testing if the patches (namely the patched NMI vector) are applied to memory or not.

PAR BIOS Versions
```
Pro Action Replay Mk1 v2.1  1992        32K CRC32=81A67556h
Pro Action Replay Mk2 v1.0  1992,93     32K CRC32=83B1D39Eh
Pro Action Replay Mk2 v1.1  1992,93,94  32K CRC32=70D6B036h
Pro Action Replay Mk3 v1.0U 1995       128K CRC32=0D7F770Ah
```
The two Mk2 versions are 99.9% same (v1.1 is only 10 bytes bigger than v1.0, major change seems to be the copyright message).
Aside from v1.0/v1.1, there are reportedly further PAR2 BIOS versions (named v2.P, v2.T, v2.H). Moreover, there's reportedly at least one localized BIOS (a german PAR3 with unknown version number).

PAR Component List
The exact component list is all unknown. Some known components are:
```
PAR1-3  3-position switch (on right edge of the cartridge)
PAR1-3  32Kbytes SRAM (probably not battery-backed)
PAR1-2  32Kbytes BIOS
PAR3    128Kbytes BIOS (with modernized GUI and built-in "PRESET" codes)
PAR1    46pin cartridge slot (incompatible with coprocessors that use 62pins)
PAR2-3  62pin cartridge slot
PAR2-3? second Npin cartridge slot at rear side (for CIC from other region)
PAR3    two LEDs (within sticker-area on front of cartridge)
PAR1-3  whatever logic chip(s)
```

SNES Cart Cheat Devices - X-Terminator & Game Wizard

Pro Action Replay (PAR1) clone
Wide parts of the X-Terminator BIOS are copied 1:1 from a disassembled PAR1 BIOS. The similarities begin at the entrypoint (with some entirely useless writes to 2140h-2143h), and go as far as using the ASCII characters "W H B ." as default dummy data values for the 4 codes (the initials of the PAR1 programmer W.H.BECKETT). There are some differences to the original PAR1: The hardware and I/O ports are a custom design, the GUI does resemble the PAR2 rather than the PAR1, and english words like "relation" are translated to odd expressions like "differentship". Nonetheless, the thing was called back (in some countries at least), presumably due to all too obvious copyright violations.

I/O Ports & Memory Map
```
X-Terminator 1         X-Terminator 2
00FFE8h.W              00FFEAh.W            ;map BIOS (by writing any value)
00FFE9h.W              00FFEBh.W            ;map GAME (by writing any value)
00FFEAh.R (NMI read)   00FFEAh.R (NMI read) ;map BIOS/GAME (switch-selection)
008000h-00FFFFh        008000h-00FFFFh      ;BIOS (32Kbytes)
N/A                    028000h-02FFFFh      ;Expansion ROM 32Kbytes
00,02,04,06:6000-7FFF  00-1F:02C00-2FFF     ;SRAM (32Kbytes)
```
Note: Both BIOS versions are confusingly using 16bit writes to the I/O ports in some cases; the LSB-write to [addr+0] has no effect (or lasts only for 1 cpu cycle), the MSB-write to [addr+1] is the relevant part.
The uncommon SRAM mapping in EVEN banks at 6000h-7FFFh was cloned from PAR. The later mapping to 2C00h-2FFFh was probably invented for compatibility with HiROM games that use 6000h-7FFFh for their own SRAM (or possibly just to look less like a PAR clone).
Aside from NMI, the X-Terminator 2 is also using IRQ vectors (though unknown if they are used only during BIOS execution or also during GAME execution, in latter case reads from FFEEh would probably also trigger memory mapping).

Game Wizard (by Innovation)
The Game Wizard seems to be a rebadged X-Terminator. Unknown if the I/O addresses are same as for X-Terminator 1 or 2.

BIOS Versions
There are at least two versions:
```
X-Terminator    1993 (english)  (CRC32=243C4A53h) (no built-in codes)
X-Terminator 2  19xx (japanese) (CRC32=5F75CE9Eh) (codes for 307 games)
```
There should be probably also a separate version for Game Wizard. And, considering that the BIOS is stored on ERPOM, there might be many further versions & revisions.
Cartridge header is FFh-filled (except for exception vectors), BIOS is 32Kbytes LoROM.

X-Terminator Expansion ROMs
There have been at least 5 expansion cartridges released:
```
Parame ROM Cassette Vol 1-5 (by Game Tech)
```
The cartridges contain 256Kbytes LoROM, and they can be used in two ways:

As normal executable (via normal ROM header at ROM-offset 7FC0h aka SNES address 00FFC0h), or as cheat-code database extension for the X-Terminator 2 (via a special ROM header at ROM-offset 10000h aka SNES address 028000h).

```
028000h - ID "FU O9149" (aka "UFO 1994" with each 2 bytes swapped)
028008h - Boot callback (usually containing a RETF opcode)
028010h - List of 16bit pointers (80xxh-FFFFh), terminated by 0000h
```

The 16bit pointers do address following structures (in ROM bank 02h):

```
2   checksum (MSB,LSB) taken from GAME cartridge ROM header [FFDCh]
1   number of following 5-byte codes (N)
5*N codes (MID,MSB,DTA,LSB,TYPE)  ;TYPE=predefined description (00h..23h)
```

Unknown how the cartridges are intended to be connected (between X-Terminator and Game cartridge... or maybe to a separate expansion slot).

Super UFO Copier

The Super UFO copiers are somehow closely related to X-Terminator (probably both made by the same company). X-Terminator codes are supported by various Super UFO versions. Later Super UFO versions also include/support Parame expansion ROMs:

```
Super UFO Pro-8 V8.8c BIOS
```

This versions seems to detect "FU O9149" IDs (ie. Parame carts), moreover, it seems to include it's own "FU O9149" ID (but, strangely, at 048000h instead of 028000h, so the X-Terminator won't find it?).

X-Terminator Chipset (whatever X-Terminator version)

```
Goldstar GM76C256ALL-70 (32Kbytes SRAM, not battery-backed)
D27256 (32Kbytes UV-Eraseable EPROM)
two logic chips & two PALs or so (part numbers not legible on existing photo)
3-position switch (on PCB solder-side) (SCAN/NORMAL/ACTION)
two cartridge slots (for PAL and NTSC cartridges or so)
```

SNES Cart Cheat Devices - Game Saver

The Game Saver from Nakitek allows to load/save snapshots of (most of) the SNES memory and I/O ports.

Game Saver Controls (works with joypad in port 1 only)

```
L+R       upon boot --> test screen / version number
L+R+START upon boot --> toggle slow DRAM checksumming on/off
SELECT    in title  --> enter revival codes
R+SELECT  in game   --> save state
L+SELECT  in game   --> load state
R+START   in game   --> toggle slow motion on/off  ;\one of these keeps
L+START   in game   --> toggle slow motion on/off  ;/HDMA enabled (or so)
```

Missing Save Data

The SNES cannot directly access the APU, so APU RAM, DSP I/O Ports, and SPC700 registers aren't saved. The WRAM address (2181h-2183h) isn't saved. The VRAM/OAM/CGRAM addresses are saved (but may have wrong values since the autoincrement isn't handled). Any coprocessor I/O ports or cartridge SRAM aren't saved.

Hardware Versions (Game Saver and Game Saver+)

The original Game Saver didn't have any power supply, which made (and still makes) it the most controverse SNES add-on: Some people just like it, other people are crying tears because they don't understand why the DRAM isn't battery-backed.
This has led to the creation of the Game Saver Plus - a surreal product that <does> use battery-backed DRAM (according to the booklet, where it is called "portability" feature, six new AA batteries last 8-10 hours). Aside from batteries, the Game Saver Plus is powered via the 9V DC supply of NTSC-SNES consoles (even when the console itself is switched off). That's allowing to switch off the SNES during supper in order to "save" energy (though after some weeks, the permanently powered DRAM may negate that energy "saving" effect).

BIOS Versions

There seem to be several BIOS versions (DDMMYY formatted date and version number are shown in the test screen). Known versions are:

```
Game Saver v1.3 (19xx)
Game Saver v1.7 (31 Jul 1995)
```

Unknown if Game Saver & Game Saver Plus use different BIOSes. Unknown if any new/changed I/O ports were invented alongside with the BIOS versions.

Game Saver Memory and I/O Map

```
002100h-0021xxh PPU ports (logged at 2081xxh) (or at 2080xxh on 2nd write)
004200h-0042xxh CPU ports (logged at 2082xxh)
008000h-00FFFFh BIOS ROM 32Kbytes
0080xFh         Switch to GAME mapping (upon opcodes that end at 80xFh)
00FFEAh         Switch to BIOS mapping (upon NMI execution; when enabled)
108000h-108001h I/O - First/second write flags for write-twice PPU ports
108002h-108003h I/O - Exception Mode/Status (bit0-1=BRK, bit2=NMI)
208000h-2087FFh SRAM 2Kbytes (includes auto-logged writes to PPU/CPU ports)
400000h-73FFFFh DRAM 256Kbytes (for saving WRAM/VRAM/OAM/CGRAM, CPU/DMA regs)
808000h-80FFFFh GAME ROM (even while BIOS mapping is enabled)
```

The Game Saver can trap BRK or NMI exceptions. Of which, BIOS v1.7 seems to use only BRKs (which are probably generated by outputting a 00h opcode in response to joypad access, ie. [4218h] reads and [4017h]=01h writes).

Game Saver Revival Codes

The 5-digit "Revival Codes" are used to improve compatibility with different games. Most commonly used are 2xxxx codes, which cause a byte in WRAM to be left unchanged when loading data (probably in order to keep the Main CPU aware of the state of the APU). The Code Format is:

```
00000-0FFFF Blank (no action) (shown as "XXXXX" in GUI)
10000-1FFFF Exception Mode (value for [108002]) (not used for any games)
20000-3FFFF Preserve WRAM byte at 7E0000-7FFFFF (used for most games)
40000-4FFFF PPU write-twice related     (used only for Starfox/Star Wing)
50000-5FFFF PPU write-twice related     (not used for any games)
60000-6FFFF Reserved (no action)        (not used for any games)
70000-7FFFF Select special BRK handler  (used only for Aero the Acro Bat)
80000-9FFFF Preserve WRAM byte at 7E0000-7FFFFF and pass it to 2140h ;\not
A0000-BFFFF Preserve WRAM byte at 7E0000-7FFFFF and pass it to 2141h ; used
C0000-DFFFF Preserve WRAM byte at 7E0000-7FFFFF and pass it to 2142h ; by any
E0000-FFFFF Preserve WRAM byte at 7E0000-7FFFFF and pass it to 2143h ;/games
```

Codes (and updates) have been available as print-outs from Nakitek (the list from 1995 contains codes for around 200 games; to be entered when pressing SELECT in title screen). Moreover some (or all) BIOSes contain automatically applied built-in codes (via checksumming portions of the game ROM header). The v1.7 BIOS contains 284 codes (however, the code list does (maybe accidently) contain an entry with NULL checksum, which causes the last 108 codes to be ignored).

Component List (Game Saver Plus)

```
24pin SRAM (2Kbytes) (probably used only because DRAM is too slow for I/O)
28pin ROM/EPROM (32Kbytes)
62pin cartridge slot (on rear side of device)
14pin eight DRAM chips (256Kbytes in total)
Xpin huge chip (whatever logic)
3pin 7805 or so (for turning much of the 9 volts into heat)
2pin oscillator (20.000MHz) (for DRAM refresh generator when power-off)
socket/cable/plug for NTSC-SNES 9V DC supply (not PAL-SNES 9V AC supply)
battery box for six 1.5V AA batteries, battery LED, and battery switch
resistors, capacitors, and maybe diodes, transistors
```

Note
Some Copiers include a similar feature, allowing to load/save "real time saves" on floppy disks and/or temporarily in unused portions of their built-in DRAM.

SNES Cart Cheat Devices - Theory

ROM Patches
There are two possible ways for patching ROMs or ROM-images:

```
1) rewrite ROM-image in RAM once before game starts        (GF/FFE/emulators)
2) patch on ROM reading (by watching address bus)          (GG and PAR2-3)
```

Both are basically having same results, there may be some variations concerning memory mirrors (depending on how the ROM-image in RAM is mirrored, or on how the GG/PAR2-3 do decode the ROM address).

WRAM Patches
Implemented by rewriting WRAM upon NMI, variations would involve mirrors:

```
1) allow WRAM addresses 7E0000-7FFFFF                      (PAR1-3, XT1-2)
2) allow WRAM addresses 7E0000-7FFFFF and nn0000-nn1FFF    (N/A)
```

SRAM Patches
There are three possible ways for patching battery-backed SRAM:

```
1) rewrite once before game starts                         (GF)
2) patch on SRAM reading (like hardware based ROM patches) (GG and PAR2-3)
3) rewrite repeatedly on NMI execution (like WRAM patches) (N/A)
```

SRAM is usually checksummed, so SRAM patches need to be usually combined with ROM patches which do disable the checksum verification. Some devices (like Game Genie) rely on the /ROMSEL signal, and thus probably can only patch SRAM in the ROM area at 70xxxxh (but not in the Expansion area at 306xxxh).

Slow Motion Feature
Implemented by inserting delays in Vblank NMI handler. The feature can be usually configured in the BIOS menu, and/or controlled via joypad button combinations from within NMI handler.

Cheat Finders (for WRAM Patches) (eventually also for SRAM patches)
Implemented by searching selected values from within Vblank NMI handler, or more simple: from within BIOS RESET handler. The search can be enabled/disabled mechanically via switch, or in some cases, via joypad button combinations. The searched value can be configured on RESET, or in some cases, via joypad button combinations.

Game Saver (Nakitek)
Allows to save a copy of WRAM/VRAM and I/O ports (but not APU memory) in DRAM, done upon joypad button-combinations sensed within BRK/NMI exception handlers.

SNES Cart Tri-Star (aka Super 8) (allows to play NES games on the SNES)

The Tri-Star is an adaptor for playing NES games on the SNES (similar to the Super Gameboy which allows to play Gameboy games on SNES). The thing have three cartridge slots (two for western/japanese NES/Famicom cartridges, and one for SNES cartridges).
NES or SNES mode can be selected in BIOS boot menu. SNES mode does simply disable the BIOS and jump to game entrypoint at [FFFCh]. NES mode executes the games via a NOAC (NES-on-a-Chip, a black blob, which is also used in various other NES clones), in this mode, the SNES video signal is disabled, and, aside from the BIOS passing joypad data to the NES, the SNES does merely serve as power-supply for the NES.

Memory and I/O Map

```
00E000h-00FFFFh.R - BIOS ROM (8Kbytes)
00FFF0h.W - NES Joypad 1 (8bit data, transferred MSB first, 1=released)
00FFF1h.W - NES Joypad 2 (bit4-5: might be NES reset and/or whatever?)
00FFF2h.W - Enter NES Mode (switch to NES video signal or so)
00FFF3h.W - Disable BIOS and map SNES cartridge
```

Joypad I/O
In NES mode, the BIOS is reading SNES joypads once per frame (via automatic reading), and forwards the first 8bit of the SNES joypad data to the NES (accordingly, it will work only with normal joypads, not with special hardware like multitaps or lightguns). Like on japanese Famicoms, there are no Start/Select buttons transferred to joypad 2. Instead, FFF1h.Bit5/4 are set to Bit5=0/Bit4=1 in SNES mode, and to Bit5=1/Bit4=0 in NES mode (purpose is unknown, one of the bits might control NES reset signal, the other might select NES/SNES video signal, unless that part is controlled via FFF3h).

Mode Selection I/O
When starting a NES/SNES game, ports FFF2h or FFF3h are triggered by writing twice to them (probably writing any value will work, and possibly writing only once might work, too).

BIOS Versions (and chksum, shown when pressing A+X on power-up)

```
Tri-Star (C) 1993                    ;ROM CHKSUM: 187C
Tri-Star Super 8 by Innovation (C) 1995  ;ROM CHKSUM: F61E
```

Both BIOSes are 8Kbytes in size (although ROM-images are often overdumped). The versions seem to differ only by the changed copyright message. The GUI does resemble that of the X-Terminator and Super UFO (which were probably made by the same anonymous company).
A third version would have been the (unreleased) Superdeck (a similar device that has been announced by Innovation and some other companies).

Component List (Board: SFFTP_C/SFFTP_S; component/solder side)

```
82pin NOAC chip (black blob on 82pin daughterboard) (on PCB bottom side)
28pin EPROM 27C64 (8Kx8) (socketed)
16pin SNES-CIC clone (NTSC: ST10198S) (PAL: probably ST10198P) (socketed)
20pin sanded chip (probably 8bit latch for joypad 1)
```

```
20pin sanded-chip (probably 8bit latch for joypad 2)
16pin sanded-chip (probably 8bit parallel-in shift-register for joypad 1)
16pin sanded-chip (probably 8bit parallel-in shift-register for joypad 2)
16pin sanded-chip (probably analog switch for SNES/NES audio or video)
20pin sanded-chip (probably PAL for address decoding or so) (socketed)
2pin  oscillator (? MHz) (for NES cpu-clock and/or NES color-clock or so)
62pin cartridge edge (SNES) (on PCB bottom side)
12pin cartridge edge (A/V MultiOut) (to TV set) (on PCB rear side)
62pin cartridge slot (SNES)
60pin cartridge slot (Famicom) (japanense NES)
72pin cartridge slot (NES) (non-japanense NES)
6pin  socket for three shielded wires (Composite & Stereo Audio in from SNES)
TV Modulator (not installed on all boards)
four transistors, plus some resistors & capacitors
```

SNES Cart Pirate X-in-1 Multicarts (1)

There are several X-in-1 Multicarts, all containing the same type of text based GUI, and thus probably all made by the same company.

Cartridge Header
The first 4 bytes of the title string at FFC0h do usually (or always) contain values 5C,xx,xx,80 (a "JMP FAR 80xxxxh" opcode, which jumps to the GAME entrypoint). The next 4 title bytes are sometimes containing another JMP FAR opcode, the rest of the header is unmodified header of the first game; except that [FFFCh] contains the MENU entrypoint.

ROM-images
ROM-images found in the internet are usually incomplete dumps, containing only the first 4MBytes (clipped to the maximum size for normal unmapped LoROM games), or only the first game (clipped to the ROM size entry of the 1st game header). Whilst, the actual multicarts are usually 8MBytes in size (there's one 4Mbyte cartridge, which is actually fully dumped).

ROM Size
Most cartridges seem to contain 8Mbyte ROMs. There is one 4MByte cartridge. And there's one cartridge that contains an 8Kbyte EPROM (plus unknown amount of ROM).

LoROM/HiROM
Most games seem to be LoROM. Eventually "Donkey Kong Land 3" is HiROM? Unknown if HiROM banks can be also accessed (dumped) in LoROM mode, and if so, unknown how they are ordered; with/without 32Kbyte interleave...?

SRAM
According to photos, most or all X-in-1 carts do not contain any SRAM. Though some might do so?

DSPn
According to photos, most or all X-in-1 carts do not contain any DSP chips. Though the "Super 11 in 1" cartridge with "Top Gear 3000" seems to require a DSP4 clone?

Port FFFFxxh

```
A0-A3 Bank Number bit0-3 (base offset in 256Kbyte units)
A4    Bank Number bit4 (or always one in "1997 New 7 in 1")
A5    Always 0         (or Bank bit4 in "1997 New 7 in 1")
A6    Varies (always 0, or always 1, or HiROM-flag in "Super 7 in 1")
A7    Always 1 (maybe locks further access to the I/O port)
```

The bank number is somehow merged with the SNES address. As for somehow: This may be ORed, XORed, or even ADDed - in most cases OR/XOR/ADD should give the same result; in case of "1997 New 7 in 1" it looks as if it's XORed(?)
The special meaning of A4-A5 can be detected by sensing MOV [FFFFnn],A opcodes (rather than normal MOV [FFFF00+x],A opcodes).
The special meaning of A6 can be detected by checking if the selected bank contains a HiROM-header.

Port 6FFFxxh
Unknown. Some games write to both FFFFxxh and 6FFFxxh (using same data & address LSBs for both ports). Maybe the ROM bank address changed to 6FFFxxh on newer boards, and FFFFxxh was kept in there for backwards compatibility. Or maybe 6FFFxxh controls SRAM mapping instead ROM mapping?

X-in-1 Cartridges

Title	FFFFxx	6FFFxx	Size/Notes
8 in 1 and 10 in 1	C0-DF	N/A	8MB (8 big games + 10 mini games?)
1997 New 7 in 1	D0-DF,F0-FF	N/A	? MB
Super 5 in 1	80-9F	80-9F	8MB
Super 6 in 1	80-8F	N/A	4MB
Super 7 in 1	80-8F,D0	80-8F,D0	8MB? (mario all stars + 3 games)
Super 11 in 1	80-9F	N/A	8MB+DSP4 ?

Chipset 7-in-1 (Board: SSF-07, REV.1)

```
U1 16pin CIVIC CT6911 (CIC clone)
U2 16pin 74LS13x or so (not legible on photo)
U3 16pin whatever      (not legible on photo)
U4 14pin 74LS02 or so  (not legible on photo)
U5 black blob
U6 black blob
```

Chipset 8-in-1 (Board: MM32-2)

```
U  20pin iCT PEEL18CV8P-25
U  16pin 93C26 A60841.1 9312 (CIC clone)
U  42pin 56C001 12533A-A 89315
U  42pin 56C005-4X 12534A-A 89317
```

Chipset 8-in-1 (Board: NES40M, 20045)

```
U  16pin CIVIC 74LS13 (CIC clone)
U  16pin not installed
U  28pin 27C64Q EPROM (8Kx8)
U  20pin iCT PEEL18CV8P-25
U  42pin JM62301
U  42pin JM62305
```

SNES Cart Pirate X-in-1 Multicarts (2)

There's at least one korean multicart, with 20 small games stored on a relative small 1Mbyte ROM. The games are NES games ported to work on SNES, some with typical pirate mods (like removing copyright strings, or renaming the game to bizarre names).

I/O Ports

```
20xxh NES PPU left-overs (written to, but ignored by the SNES)
40xxh NES APU left-overs (written to, but ignored by the SNES)
8000h ROM Bank Size/Base
```

Port 8000h works around as so:

```
0-4  ROM Base Offset (in 32Kbyte units)
5    Unknown/unused (always zero)
6-7  ROM Bank Size (0=Used/unknown, 1=Unused/Unknown, 2=1x32K, 3=2x32K)
```

The ROM is mapped in LoROM fashion (with 1 or 2 banks of 32Kbyte).
SRAM might also exist (the photo shows some unidentified 24pin chip).

Component List

```
PCB Name: Unknown (it has one, but isn't legible on lousy photo)
32pin C20H (1Mbyte ROM)
24pin Unknown (maybe SRAM) (there is no battery visible on PCB front side)
20pin Unknown (looks like a sanded chip; presumably memory mapper)
16pin CIVIC CTxxxx? (CIC clone)
46pin Cartridge Edge Connector
```

SNES Cart Copiers

Copiers

SNES Cart Copiers - Front Fareast (Super Magicom & Super Wild Card)
SNES Cart Copiers - CCL (Supercom & Pro Fighter)
SNES Cart Copiers - Bung (Game Doctor)
SNES Cart Copiers - Super UFO
SNES Cart Copiers - Sane Ting (Super Disk Interceptor)
SNES Cart Copiers - Gamars Copier
SNES Cart Copiers - Venus (Multi Game Hunter)
SNES Cart Copiers - Others

Misc

SNES Cart Copiers - Misc

Floppy Disc Controllers

SNES Cart Copiers - Floppy Disc Controllers
SNES Cart Copiers - Floppy Disc NEC uPD765 Commands
SNES Cart Copiers - Floppy Disc FAT12 Format

BIOSes

SNES Cart Copiers - BIOSes

See also

SNES Cartridge ROM-Image Headers and File Extensions

SNES Cart Copiers - Front Fareast (Super Magicom & Super Wild Card)

Front/CCL/Clones

The Front Fareast I/O addresses are used by Front's own models, by early CCL models, and by some third-party clones:

```
Super Magicom (Front/CCL)
Super Wild Card (Front)
Supercom Pro (CCL) (later CCL models use other I/O ports)
Super Drive Pro-3 UFO (noname) (later UFO models use other I/O ports)
```

I/O Ports (in banks 00h..7Dh and 80h..FFh)

```
C000.R    FDC Flags (Bit7: MCS3201 IRQ Signal, Bit6: Drive 'Index' Signal)
            Note: Index signal is (mis-)used for Disk Insert Check
C002.W    FDC MCS3201 Drive Control Register (motor on, etc.)
C004.R    FDC MCS3201 Main Status Register
C005.RW   FDC MCS3201 Command/Data Register
C007.R    FDC MCS3201 Diagnostics Register (bit7=disk change; MCS-chip only)
C007.W    FDC MCS3201 Density Select Register (bit0-1=Transfer rate)
C008.R    Parallel Data Input (Reading this register reverses busy flag)
C008.W    Parallel Data Output (bit0-3) and DRAM/SRAM mapping (bit0-1)
            Bit 0: 0=LoROM/Mode 20, 1=HiROM/Mode 21 (DRAM Mapping)
            Bit 1: 0=LoROM/Mode  1, 1=HiROM/Mode  2 (SRAM Mapping)
C009.R    Parallel Port Busy Flag, Bit 7 (older EP1810 Version) (Altera chip)
C000.R    Parallel Port Busy Flag, Bit 5 (newer FC9203 Version) (FRONT chip)
C00A-C00F Unused (mirrors of C008h-C009h)
C010-DFFF Unused (mirrors of C000h-C00Fh)
```

Below E000h-E00Dh are triggered by writing any value

```
E000.W    Memory Page 0  ;\Select an 8Kbyte page, CART/DRAM/SRAM address is:
E001.W    Memory Page 1  ;    SNES address AND 1FFFh      ;lower bits
E002.W    Memory Page 2  ;    +Selected Page * 2000h      ;upper bits
E003.W    Memory Page 3  ;/   +SNES address AND FF0000h   ;bank number
E004.W    Set System Mode 0 (BIOS Mode)            (with all I/O enabled)
E005.W    Set System Mode 1 (Play Cartridge)       (with all I/O disabled)
E006.W    Set System Mode 2 (Cartridge Emulation 1) (with E004-E007 kept on)
E007.W    Set System Mode 3 (Cartridge Emulation 2) (with all I/O disabled)
E008.W    Select 44256 DRAM Type  (for 2,4,6,8 Mega DRAM Card)
E009.W    Select 441000 DRAM Type (for 8,16,24,32 Mega DRAM Card)
E00C.W    BIOS Mode:CART at A000-BFFF, DRAM Mode:DRAM in bank 20-5F/A0-DF
E00D.W    BIOS Mode:SRAM at A000-BFFF, DRAM Mode:CART in bank 20-5F/A0-DF
```

Later Wild Card DX models have various extra ports, eg. E0FDh, E083h, C108h

Ports C00xh seem to be used by models up to DX and DX96.
In DX2, Ports C00xh seem to be moved to CF8xh/DF8xh.

System Mode 0 (BIOS Mode) (selected via E004h)
```
bb2000-bb3FFF RW: SRAM or CART (E00C/E00D)  bb-40-7D,C0-FF ;\8K page via
bb8000-bb9FFF RW: DRAM                      bb-00-7D,80-FF ; E000-E003
bbA000-bbBFFF RW: SRAM or CART (E00C/E00D)  bb=00-7D,80-FF ;/
bbC000-bbC00x RW: I/O Ports                 bb=00-7D,80-FF
bbE000-bbE00x W : I/O Ports                 bb=00-7D,80-FF
bbE000-bbFFFF R : BIOS ROM (8/16/256Kbytes) bb=00-1F
```

System Mode 1 (CART Mode) (selected via E005h)
```
bb0000-bbFFFF RW: CART
```

System Mode 2/3 (DRAM Modes) (selected via E006h/E007h)
```
bb0000-bb7FFF R : DRAM Mapping, bb=40-6F, C0-DF. (HiROM/Mode 21)
bb8000-bbFFFF R : DRAM Mapping, bb=00-6F, 80-DF. (AnyROM/Mode 20,21)
708000-70FFFF RW: SRAM Mode 1 Mapping.         ;<-- typically for LoROM
306000-307FFF RW: SRAM Mode 2 Mapping, Page 0. ;<-- typically for HiROM
316000-317FFF RW: SRAM Mode 2 Mapping, Page 1. ;\extra banks for HiROM
326000-327FFF RW: SRAM Mode 2 Mapping, Page 2. ; (do any 'real' cartridges
336000-337FFF RW: SRAM Mode 2 Mapping, Page 3. ;/do actually have that?)
```
DRAM mapping (LoROM/HiROM), and corresponding SRAM mapping are selected via (sharing) Bit0-1 of the Parallel Data Output (Port C008h.W)
HiROM/Mode 21:
```
Even DRAM Bank is mapped to bb0000-bb7FFF.
Odd DRAM Bank is mapped to  bb8000-bbFFFF.
```
Optionally, banks 20-5F and A0-DF can be mapped to CART instead of DRAM (via E00Dh), probably intended to allow ROM-images in DRAM to access DSP chips in CART.

BIOS Notes
Observe that the BIOS is divided into 8Kbyte banks (so, the exception vectors are at offset 1Fxxh in the ROM-image) (however, there are some overdumped ROM-images that contain 24K padding prior to each 8K ROM-bank, ie. with LoROM mapping style exception vectors at offset 7Fxxh). Aside from the exception vectors, there isn't any title, nor other valid cartridge header entries.
Note: Unlike most SNES programs, Magicom & Wild Card (until v1.8) BIOSes are running in 6502 emulation mode (with E=1), rather than 65C816 mode (with E=0).

Parallel Port Protocol (on SNES side)
Data is received via 8bit data register, and sent via 4bit "status" register (which is seen as status on PC side). Strobe/busy aren't clearly documented in official Front specs; probably, Busy gets set automatically when sensing Strobe (from PC side), and gets cleared automatically when reading Data from Port C008h (on SNES side).

Parallel Port Protocol (on PC side)
Byte Output Procedure:
```
Wait Busy Bit = 1           ;Status  PC Port 379h/279h/3BDh.Bit7
Write One Byte              ;Data    PC Port 378h/278h/3BCh.Bit0-7
Reverse Strobe Bit          ;Control PC Port 37Ah/27Ah/3BEh.Bit0
```
Byte Input Procedure:
```
Wait Busy Bit = 0           ;Status  PC Port 379h/279h/3BDh.Bit7
Read Low 4 Bits of Byte     ;Status  PC Port 379h/279h/3BDh.Bit3-6
Reverse Strobe Bit          ;Control PC Port 37Ah/27Ah/3BEh.Bit0
Wait Busy Bit = 0           ;Status  PC Port 379h/279h/3BDh.Bit7
Read High 4 Bits of Byte    ;Status  PC Port 379h/279h/3BDh.Bit3-6
Reverse Strobe Bit          ;Control PC Port 37Ah/27Ah/3BEh.Bit0
```
Receiving 4bit units via status line is done for compatibility with old one-directional PC parallel (printer) ports.
Unknown if "Wait Busy Bit = X" means to wait while-or-until Bit=X?

Parallel Port Command Format
Commands are 9-bytes in length, sent from PC side.
```
00h 3   ID (D5h,AAh,96h)
03h 1   Command Code (00h-01h, or 04h-06h)
04h 2   Address (LSB,MSB)
06h 2   Length (LSB,MSB)
08h 1   Checksum (81h XORed by Bytes 03h..07h)
Followed by <Length> bytes of data (upload/download commands only)
```
Commands can be:
```
Command 00h : Download Data (using page:address,length)   ;to-or-from PC?
Command 01h : Upload Data (using page:address,length)     ;from-or-to PC?
Command 04h : Force SFC Program to JMP (to address... plus page/bank?)
Command 05h : Select 8Kbyte Memory Page Number (using address)
Command 06h : Sub Function (address: 0=InitialDevice: 1=ExecDRAM, 2=ExecCART)
```
InitialDevice does probably reset the BIOS? ExecDRAM allows to run the uploaded ROM-image, but unknow how to select LoROM/HiROM mode, or is it automatically done by examining the uploaded cartridge header. The usage of the 16bit address isn't quite clear: The lower 13bit are somehow combined with the 8Kbyte page number, the upper 3bit might be used to select DRAM/SRAM?

Super Magicom V3H - BIOS upgrade
This ROM-image is a Magicom BIOS upgrade, and it's a pain in the ass:
The upgrade works ONLY as ROM-image (ie. must be loaded to DRAM), and does NOT work as real ROM (ie. cannot be burned to EPROM), the reason is that it doesn't include a character set (and uses that from the original Magicom BIOS at E000h).
The upgrade isn't compatible with the parallel port (the original BIOS relocates parallel port code from ROM to WRAM, and the upgrade relocates itself from DRAM to WRAM - but still expects the parallel port code in the same WRAM location and crashes when Busy-bit gets set).
The upgrade exists as 32Kbyte or 32.5Kbyte ROM-image (an 8K upgrade, 24K garbage with entrypoint at end of garbage, plus 0.5K extra garbage), emulators and other tools will be typically interprete the 32.5K file as 32Kbyte ROM with 512-byte header (which is NOT correct in that special case).
The upgrade exists in two variants: One using the standard Front Fareast I/O addresses, one using the I/O addresses at C000h-C00Fh re-ordered as so:
```
8000h-FFFFh RW DRAM-mode: DRAM (containing the Magicom V3H upgrade)
E000h-FFFFh R  BIOS-mode: BIOS (containing the Character set)
C000h       W  DRAM bank mapped to 8000h? (set to 00,20,40 upon DRAM detect)
C001h       W  Memory Control?
C002h       -  Unused
C003h       R  Parallel Port Busy (bit7) (when set: crashes the V3H upgrade)
C004h-C008h -  Unused
C009h       R  Status (bit7=ready?,bit5=busy/timeout?)
C00Ah       -  FDC Unused
C00Bh       W  FDC Motor Control (set to 00h,29h,2Dh)
```

```
C00Ch       RW FDC Command/Data
C00Dh       R  FDC Main Status
C00Eh       W  FDC Transfer Rate? (set to 00h,01h,02h or so)
C00Fh       -  FDC Unused
E004h       W  Map BIOS ROM (instead V3H upgrade) ;\
E006h-E007h W  Something on/off                   ; seems to be same/similar
E008h-E009h W  Something on/off                   ; as Front-like I/O ports
E00Ch-E00Dh W  Something on/off                   ;/
```

Unknown which hardware uses that re-ordered addresses. Note: The V3H version with re-ordered I/O addresses does also contain different 24K garbage (sorts of as if it were created from original source code, rather than just patched?).

Component List - Super Magicom Plus

```
U1  24pin  DRAM (onboard)
U2  20pin  SN74LS245N (8-bit 3-state transceiver)
U3  24pin  DRAM (onboard)
U4  24pin  DRAM (onboard)
U5  24pin  DRAM (onboard)
U6  28pin  27C128-25 EPROM (16Kx8)
U7  68pin  MCCS3201FN (=MCS3201FN without double-C) (disc controller)
U8  100pin ?
U9  28pin  HM62256 (SRAM 32Kx8)
U10 20pin  ?
U11 14pin  ?  (does not exist in later versions?)
U12 20pin  AMI 16CVB8PC-25
U13 20pin  AMI 16CVB8PC-25
U14 16pin  ST10198S (newer version only) (mounted on top of U10 in old ver)
BT1 2pin   ?
J1  25pin  DB-25 parallel port
J2? 25pin  DB-25 external floppy (not installed)
J3  40pin  DRAM expansion board
J4  34pin  internal floppy (flat cable)
J5  26pin  (not installed)
J6  4pin   floppy power supply
J7  62pin  cartridge edge
J8  62pin  cartridge slot
J9  12pin  jumpers (I:I:I: or :I:I:I) (enable internal or external CIC)
Y1  2pin   24.000 MHz
```

Component List - Super Wild Card DX (AH-558001-02 Made in Japan 94.8.23)

```
U1  100pin CPU      FRONT FC9203 HG62E22926F9 (or so) (SMD)
U2   28pin S-RAM    NEC D43256AC-10L (uPD43256AC) (SRAM, 32Kx8)
U3   20pin          SN74LS245 (to parallel port) (8-bit 3-state transceiver)
U4    ?pin PAL-2    L GAL20V84 25LP
U5   20pin          SN74LS...? (or so)
U6   32pin BIOS-ROM BIOS
U7   16pin U7       SN74LS139AN (decoder/demultiplexer)
U8   14pin U8       SN74LS125AN (quad 3-state buffer)
U9   44pin          GoldStar GM82C765B (SMD) (floppy disc controller)
U10  16pin DECODER  SNC4011 (or so)
U11  20pin PAL-3    iCT PEEL17CV8P CTN24053
U12?  3pin 7805H    voltage regulator
U13  20pin PAL-1    AMI
X1    2pin 16MHZ    16.000 MHz
CN?   2pin AC/DC-IN power supply input
CN?   4pin ..POW    power supply to internal disc drive (only 2pin connected)
CN2  62pin          female cartridge slot
CN3? 46pin RAM-SLOT to DRAM daughterboard (only 40pin used on remote side)
CN5  25pin PC-I/F   DB-25 parallel port
CN6  34pin FDD-I/F  cable to internal disc drive
CN01 34pin          goes to one 1st of male 62pin cartridge edge
CN02 34pin          goes to one 2nd of male 62pin cartridge edge
SW1   3pin RESET-SW reset switch/button or so, for whatever purpose
DB1   4pin          AC-DC converter
BT1   2pin          3V battery
J1   12pin          jumpers (near cartridge slot)
J2   20pin          jumpers (near cartridge slot)
J3    2pin          jumper (near power-input)
J4    2pin          jumper (near power-input)
J5    2pin          jumper (near power-input)
DRAM Daughterboard:
U1,U2,U7,U8   16pin  ST T74LS139B (decoder/demultiplexer) (four pieces)
U3-U6,U9-U10  28pin  NEC D424900G5 (or so) (six pieces) (SMD)
U11-U12       28pin  M5M44800ATP           (two pieces) (SMD)
```

Component List - Supercom Pro (SP3200) (dated around 1992)

(probably uses Front-like I/O)

```
U1  20pin  SN74LS245N (to parallel port) (8-bit 3-state transceiver)
U2  16pin  HD74LS174 (to parallel port?)
U4  68pin  MCCS3201FN (=MCS3201FN without double-C) (floppy disc controller)
U?  68pin  Altera EP1810LC-45 D9219
U?  28pin  EPROM
U?  28pin  SRAM Winbond W24256-10L 9149
U7  20pin  SN74LS245N (8-bit 3-state transceiver)
U8  16pin  not installed
U?  20pin  modded (?) chip (soldered on cart-edge connector at bottom side)
J1  25pin  DB-25 parallel port
J2  25pin  DB-25 external floppy disc connector
J3  40pin  to DRAM daughterboard
J4  34pin  not installed (internal floppy disc connector)
J5  62pin  cartridge edge
J6  62pin  cartridge slot
Y1  2pin   24.000 MHz
BT1 2pin   VARTA Ni/Cd, 3.6V 60mA, 14h 6mA (recharge-able & acid-leaking)
```

Component List - SMD800 Super Magic Drive (requires SNES-to-Genesis adaptor)

```
U1  20pin  SN74LS245N (to parallel port?) (8-bit 3-state transceiver)
U2  16pin  SN74LS174 (to parallel port?)
U3  20pin  SN74HC245P (8-bit 3-state transceiver)
U4  20pin  SN74HC245P (8-bit 3-state transceiver)
U5? 68pin  MCS3201FN (floppy disk controller)
```

```
  U6  20pin  SN74HC245P (8-bit 3-state transceiver)
  U   28pin  27C64A-15 (EPROM, 8Kx8) (with Genesis Z80 code, non-SNES code)
  U   28pin  HY62256ALP-10 (SRAM, 32Kx8)
  U     pin  Altera EP1810LC-45
  U10 16pin  MC74HC157 (decoder/demultiplexer)
  U11 16pin  MC74HC157 (decoder/demultiplexer)
  U12 14pin  xxxx
  J   25pin  DB-25 parallel port
  J2  25pin  DB-25 external floppy
  J3  40pin  internal floppy (not installed) (likely only 34pins of 40pin used)
  J   64pin  cartridge edge (genesis)
  J   64pin  cartridge slot (genesis)
  J   40pin  to DRAM daughterboard
  Y   2pin   oscillator
  BT  2pin   VARTA Ni/Cd, 3.6V 60mA, 14h 6mA (recharge-able & acid-leaking)
 DRAM Daughterboard:
  U1  20pin  HY514400J-70 (DRAM)
  U2  20pin  HY514400J-70 (DRAM)
  U3  20pin  HY514400J-70 (DRAM)
  U4  20pin  HY514400J-70 (DRAM)
  U5  14pin  74LS08 (quad 2-input AND gates)
  U6  16pin  HD74LS157P (decoder/demultiplexer)
  U7  14pin  74LS08 (quad 2-input AND gates)
  CN1 40pin  connector to mainboard
 Super Magicom-Drive (SNES-to-Genesis adaptor for above):
  xxx        components unknown
```

SNES Cart Copiers - CCL (Supercom & Pro Fighter)

Below is for Supercom Partner & Pro Fighter models from CCL (China Coach Limited). See the Front Fareast chapter for their earlier Super Magicom models (which were produced by Front & CCL), and also for Supercom Pro 2 (which was made by CCL alone, but still used the Front-like I/O ports).

Pro Fighter 1993 by H.K. / Supercom Partner A
Ports are somewhat based on the Front design (BIOS is expanded from 8K at E000h-FFFFh to 16K at C000h-FFFFh, accordingly FDC Ports C000h-C007h are moved to 2800h-2807h, and Parallel Port Ports C008h-C009h are simply removed. Ports at E00xh are somehow changed, but might be still similar to the Front design (?)

```
  2800.R   FDC MCS General Purpose Input (bit7,bit6 used)
  2802.W   FDC MCS Motor Control (set to 00h,29h,2Dh)
  2804.R   FDC MCS Main Status
  2805.RW  FDC MCS Command/Data Status
  2807.W   FDC MCS Transfer Rate/Density (set to 0..3)
  Below 2808-2810 only in newer "Pro Fighter Q"
   2808.R   Parallel Port Data (bit0-7)
   2809.W   Parallel Port Data (4bit or 8bit?)
   2810.R   Parallel Port Busy (bit5)
    (there seem to be 4bit & 8bit parallel port modes supported, one of them
    also WRITING to 2808h, and in some cases reading "FDC" register 2800 looks
    also parallel port DATA and/or BUSY related)
  Again changed for Double Pro Fighter
   2803.R   Parallel Port Busy (bit7)
   2808.R   Parallel Port Data (bit0-7)
   2809.W   Parallel Port Data (4bit or 8bit?)
   2804 =FDC DATA   ;\swapped ! (unlike older "non-double" models)
   2805 =FDC STAT   ;/
   280x =other ports in this region may be changed, too ?
   004800    ROM (from offset 8800-9FFF) (contains program code)
   014800    ROM (from offset A800-BFFF) (contains character set)
   E00x
   E800+x
   Note: Having BIOS portions mapped to the fast 3.58MHz region at 4800h-5FFFh
         was probably done unintentionally; this would require 120ns EPROMs,
         whilst some Double Pro Fighter boards are fitted with 200ns EPROMs
         (which are stable at 2.68MHz only, and may cause crashes, or charset
         glitches in this case)
    Double Pro Fighter BIOS is 64Kbytes:
      0000-3FFF  Genesis/Z80 BIOS
      4000-7FFF  Same content as 0000-3FFF
      8000-87FF  Unused (zerofilled)
      8800-9FFF  SNES BIOS (6K mapped to 004800-005FFF)
      A000-A7FF  Unused (zerofilled)
      A800-BFFF  SNES BIOS (6K mapped to 014800-015FFF)
      C000-FFFF  SNES BIOS (16K mapped to 00C000-00FFFF)
  7000.R
  A000.RW              ;7000-related
  C000-FFFF.R BIOS ROM (16Kbytes)
  E002.W   set to 00h   ;7000-related
  E003.W   set to BFh ;then compares BFFD with BFFC,BFFA,BFFB,BFEA,BFEB
  E00C.W   set to 00h   ;7000-related
  E00E.W   set to E0h
  008000.RW   DRAM detection?
  208000.RW   DRAM detection?
  408000.RW   DRAM detection?
  608000.RW   DRAM detection?
```

SNES Cart Copiers - Bung (Game Doctor)

Game Doctor SF7

Memory Map (in BIOS mode)

```
  00:8000-807F     I/O Ports
  00:8080-FFFF     BIOS ROM (1st 32kBytes)
  01:8000-FFFF     BIOS ROM (2nd 32kBytes) (if any)
  02:8000-FFFF     unused
  03:8000-FFFF     unused
  04:8000-FFFF     SRAM for game positions (32Kbyte)
```

```
  06:8000-FFFF     SRAM for copier settings (4kByte)
  07:8000-FFFF     DRAM for ROM-image (32Kbyte page, selected via Port 8030h)
  08-7D:8000-FFFF  Mirror of above banks 00-07
  80-FF:8000-FFFF  Mirror of above banks 00-07 or Cartridge banks 00-7F/80-FF

  FFBFh compared to FFh ?
```

I/O Ports (in BIOS mode) in bank 00h

```
  8000h-800Fh RW 512Kbyte DRAM chunk, mapped to upper 32Kbyte of Bank 0xh-Fxh
  8010h-8013h RW 512Kbyte DRAM chunk, mapped to lower 32Kbyte of Bank 4xh-7xh
  8014h-8017h RW 512Kbyte DRAM chunk, mapped to lower 32Kbyte of Bank Cxh-Fxh
  8018h-8019h W  SRAM Flags (bit0-15=Enable SRAM at 6000-7000 in banks 0xh-Fxh)
  8018h       R  bit1 = realtime.$4016.bit0, read bit7 = ? , bit = ?
  8018h.R  Flags (bit7/6 FDC IRQ?, and more)
  8019h       R  bit1 = ?
  801Ah       R  realtime.word, latch settings for double write word registers
  801Ah       W  write ?
  801Bh       W  write ?
  801Dh       W  BIOS mode mapping: changes what is mapped into banks $80-$FF
                   only bit0-bit1 seem to matter
                   0 = use cartridge banks $00-$7F
                   1 = use cartridge banks $80-$FF
                   2 = mirror banks $00-$7F (BIOS regs and all?)
                   3 = mirror banks $00-$7F (BIOS regs and all?)
  801Eh write ?
  __Floppy Disc__
  8020h       R  FDC Main Status
  8021h       RW FDC Command/Data
  8022h       W  FDC Transfer Rate/Density (?) (set to 00h,01h)
  8023h       -  FDC Unused
  8024h       W  FDC Motor Control (set to 00h,08h,0Ch,1Ch,2Dh)
  8025h-8027h -  FDC Unused
  8028h       W  set to same value (ANY VALUE?) as 8022/8029)
  8029h       W  set to same value (ANY VALUE?) as 8029)
  802Ah       W  set to 01-then-00 (once) (thereafter do sth to 8022)
  802Bh       W  set to 01h during FDC COMMAND-BYTEs (else to 00h) (maybe LED?)
  __Parallel Port__
  802Ch       RW Parallel Port Data Lines
  802Dh       RW Parallel Port Status Lines
  802Eh       RW Parallel Port Control Lines
  802Fh       W  Parallel Port? Unknown (set to 00h,01h) (data direction?)
  802Fh       R  Parallel Port? Unused (reads same as $00802D)
  __Memory__
  8030h-8031h W  Select 32Kbyte-DRAM-Page (0000h..01FFh) mapped to 078000h
  8030h-803Dh R  this is a 7 word table?? (gotten from code at 80/AE80)
  8040h-805Fh R  read same as 802Dh (uh, but, some are used for sth else?)
  8040h       R  used, parallel port related (or other mainboard version?)
  8043h       W  used, parallel port related (or other mainboard version?)
  8060h-807Fh R  read = FFh
  80xFh          any access to 0080xFh (x=8..F) switches to cartridge mode
```

802Dh - Parallel Port status (not direct pin reading?)

```
 read
  bit0 = /C1 (direct pin14, /AutoLF) (/Ctrl.Bit1 on PC side)
  bit1 = C2  (direct pin16, /INIT)   (Ctrl.Bit2 on PC side)
  bit2 = /C3 (direct pin17, /Select) (/Ctrl.Bit3 on PC side)
  bit3 = "write bit3"
  bit4 = "write bit4"
  bit5 = "write bit4" (uh, not bit5 here?)
  bit6 = "write bit4" (uh, not bit6 here?)
  bit7 = /S7 (direct pin11) = "write bit7" AND not "write bit0"
 write
  bit0 Enable/Disable Busy bit (0=Enable, 1=Disable) (?)
  bit3 => S3 (direct pin15, /ERR) (Stat.bit3 on PC side)
  bit4 => S4 (direct pin13, SLCT) (Stat.bit4 on PC side)
  bit5 => S5 (direct pin12, PE)   (Stat.bit5 on PC side)
  bit6 => S6 (direct pin10, /ACK) (Stat.bit6 on PC side)
  bit7 ...   (direct pin11, BUSY  (/Stat.bit7 on PC side) (ANDed with /bit0?)
```

802Eh - Parallel Port control (not direct control reg values)

```
 often write 12h-then-10h
 read/write?
  bit0 = /C1 (direct pin14, /AutoLF) (/Ctrl.Bit1 on PC side)   W
  bit1 = C2  (direct pin16, /INIT)   (Ctrl.Bit2 on PC side)    W
  bit2 = /C3 (direct pin17, /Select) (/Ctrl.Bit3 on PC side)   W
  bit3-bit6, read = bit3-bit6 of $00802D                       W
  bit7 = /C0 (direct pin1,  /STB) (/Ctrl.bit0 on PC side)      R
```

Component List - Bung Game Doctor SF6 (Board CT401)

```
  U    3pin 7805 or so
  U   40pin GoldStar xxx (=probably GM82C765B) (floppy disc controller)
  U  ???pin huge chip (200 pins or so)
  U   18pin 265111
  U   20pin 74LS744 or so (not installed)
  U   28pin SRAM or so
  U   28pin SRAM or so
  U   28pin EPROM (GDSF_6.0)
  U   14pin whatever/modded chip (wired top-down near EPROM)
  P   40pin to DRAM daughterboard 1 (2x10 male pins, 2x10 female pins)
  P   40pin to DRAM daughterboard 2 (2x10 male pins, 2x10 female pins)
  P   62pin cartridge port
  P   62pin cartridge port (on PCB back side)
  P   25pin DB-25 parallel port (on PCB back side)
  P    2pin power supply (on PCB back side)
  P    2pin floppy supply (on PCB back side)
  P   34pin floppy data
  X    2pin oscillator
```

SNES Cart Copiers - Super UFO

UFO Super Drive Pro / Super UFO

UFO3
The UFO-3 is a Front Fareast clone. BIOS is 8Kbytes mapped to E000h-FFFFh, FDC Registers are at C000h-C007h, Parallel Port at C008h-C009h, Memory Control at E000h-E00Dh. For details see Front Fareast chapter.

UFO6
I/O ports for this version are unknown.

UFO7/UFO8
```
  2184.W   ... set to 00h/0Ch/0Fh
  2185.W   ... set to 00h/0Fh
  2186.W   ... set to 00h/0Fh
  2187.W   ... set to 08h/00h/0Bh
  2188.W   ... set to 00h..0Fh or so
  2189.W   ... set to 0Fh/0Eh
  218A.W   ... set to 00h
  218B.W   ... set to 0Ah/0Fh
  218C.R   FDC Main Status Register
  218D.RW  FDC Command/Data Register (emit 03h,DFh,03h = spd/dma)(then 07h,01h)
  218E.W   FDC Motor Control (set to 00h, 29h-then-2Dh on disc access)
  218F.W   FDC Transfer Rate
  218F.R   FDC Flags (bit7=irq?,bit6=index?) (UFO8: bit5=?)
  003F68.R         warmboot flag? if A581 --> JMP 3D00
  003FD0..3FFF   cartridge header? (or copy of it?)
  003C00..003FFF SRAM 1Kbyte (BIOS settings, I/O logging?, last 32-byte OAM)
  013C00..013FFF SRAM 1Kbyte (512-byte Palette and 1st 512-byte OAM)
  008000h and up BIOS 64Kbytes (UFO7) or more 128K..256K (UFO8)
  708000h and up SRAM 32Kbytes (for game positions)
  808000h and up DRAM (variable size detected) (via calls to 9025)
```
ufo7 rom chksum calculated at 9505 (32K ROM at 8000-FFFF must sum up to 00h)
ufo8 (and maybe ufo7 too) should have 8K SRAM (ie. MORE than above 2x1K...?)

UFO6 Component List - UFO Super Drive Pro (with Pro-6 BIOS)
```
  U    3pin 7805 or so
  U    ?pin xxxx (near 7805)
  U   40pin GoldStar GM82C765B
  U   14pin xxxx
  U   20pin L GALxxxx
  U   20pin L GALxxxx
  U   20pin L GALxxxx
  U   20pin AMI xxxxx
  U   20pin L GALxxxx
  U   20pin L GALxxxx
  U   20pin L GALxxxx
  U   20pin L GALxxxx
  U   20pin LS245 (not installed, near DB-25) (8-bit 3-state transceiver)
  U   14pin HC74  (not installed, near DB-25) (dual flip-flop)
  U   16pin xxxx  (installed, near DB-25)
  U   28pin EPROM
  U   28pin Winbond W24256-10L (SRAM 32Kx8)
  U   20pin Philips PC74xxxx
  U   16pin 74LS112 (reportedly a cloned/mislabelled CIC chip)
  X    2pin oscillator (near 7805)
  BT   2pin 3V or so
  P   25pin DB-25 parallel port or so
  P    2pin power supply
  P    2pin floppy supply
  P   34pin floppy data
  P   40pin to DRAM daughterboard
  P   46pin cartridge slot (only 46 soldering points)
  P   62pin cartridge edge (has 62 soldering points, but only 46 connected?)
```

UFO8 Component List Super UFO Super Drive PRO8 (REV 7.8 2)
```
  1x 84pin Altera EPMxxxxxxx84-15
  1x 40pin GoldStar GM82C765B (DIL) (floppy disc controller)
  1x 32pin BIOS ROM/EPROM (located on PCB solder side)
  1x 28pin UM62256D-70L (SRAM 32Kx8)
  1x 28pin UT6264PC-70LL (SRAM 8Kx8)
  1x 28pin DSP chip (not installed)
  2x 24pin NN5117405BJ-60 (DRAM, two pieces, located on daughterboard)
  1x 16pin D1
  1x 14pin FT4066
  1x 14pin DSP_74HC74 (not installed) (dual flip-flop)
  1x 14pin 74LS00 or so
  1x 14pin whatever (near oscillator)
  1x 14pin 74LSxxx whatever (near PAL/NTSC jumpers)
  2x 16pin SN74HC157N (decoder/demultiplexer)
  1x 3pin  7805 or so
  1x 34pin connector/cable to internal disc drive
  2x 62pin cartridge connectors (one male, one female)
  1x 2pin  wire (supply to internal disc drive)
  1x 2pin  connector (external power supply input)
  no battery, no parallel port
```

SNES Cart Copiers - Sane Ting (Super Disk Interceptor)

The Super Disk Interceptor is a SNES copier from KL818 B.C./Sane Ting Co. Ltd., the company also made a copier for Mega Drive, called Mega Disk Interceptor.

I/O & Memory
```
  8000-9FFF memory (SRAM/DRAM or so)
  A000.W  set to 00,03-then-01, or 40,80
  A001.W  set to 00,04 (as MSB of A000) or to 04,24,08
```

```
A002.W  FDC Transfer Rate/Density (set to ([0B] XOR 1)*2)
A003.W  FDC Motor Control (set to 08,0C,1C)
A004    FDC Unused
A005.RW FDC Command/Data
A006.R  FDC Main Status
A007    FDC Unused
A008.W  set to [1802] (bit3,bit4 used)
B000-B01F ...    I/O or RAM or RegisterFile workspace?
 B000.W  set to 00
 B000.R  checked if 00h
 B001.W  set to 00
 B002.W  set to xx OR 80h
 B002.R  read and ORed with 06h
 B003.W  set to xx OR 80h OR 03h
 B003.R  bit5 isolated, ORed with 04h, then written to A001h
 B004.W  set to 00 or 00..03h
 B004.R  whatever, if (N+1)=00..03 --> written to B004 and B005
 B005.W  set to 00
 B006.RW set to [4219h] = MSB of joypad1 (?)
 B00F.W  set to [FFDC]=00h or ([FFDC] XOR 1)=01h
 B00F.R  checked if 00h (if nonzero --> WRITE PROTECT)
 B01x    ...
C000.R  dummy read within waitvblank
C001.R  dummy read within waitvblank
E000.W
E001.W
E002.W
E000-FFFF BIOS (32Kbytes, in 8Kbyte units, in banks 00h-03h)
704000
708000
```

Component List - Super Disk Interceptor (version dated around 1992)

```
U1  40pin  GoldStar GM82C765B PL (DIP) (floppy disk controller)
U2  84pin  MD1812 9211 (with socket)
U3  28pin  2xxx4A-25 (PROM, presumably 8Kx8, non-eraseable)
U4  20pin  not installed (DIP) (BANK2.3)
U4A 20pin  not installed (DIP) (BANK2.3)
U5  20pin  GoldStar GMxxxxx (SMD) (BANK0.1) (DRAM)
U5A 20pin  GoldStar GMxxxxx (SMD) (BANK0.1) (DRAM)
U6  28pin  Hyundai HY62256ALP-10 (SRAM, 32Kx8)
U7  20pin  not installed (DIP) (BANK2.3)
U7A 20pin  not installed (DIP) (BANK2.3)
U8  20pin  GoldStar GMxxxxx (SMD) (BANK0.1) (DRAM)
U8A 20pin  GoldStar GMxxxxx (SMD) (BANK0.1) (DRAM)
U9  16pin  74HC157N (decoder/demultiplexer)
U10 16pin  74HC157N (decoder/demultiplexer)
U11 20pin  HY-xxxxxx-30
U12 20pin  HY-xxxxxx-30
XTAL 2pin  16.000MHz
J   46pin  Cartridge edge (snes)
J   46pin  Cartridge slot (snes)
J   34pin  Floppy data
J    2pin  Floppy supply
BT  2pin   3.6V Battery
```

Component List - Super Disk Interceptor (version dated around 1993)

```
U   44pin  GoldStar GM82C765B PL (SMD) (floppy disk controller)
U   28pin  27C64SDM (PROM, 8Kx8, non-eraseable)
U   28pin  GoldStar GM76C256ALLFW70 (SRAM, 32Kx8)
U   20pin  HD74HC373P (8-bit 3-state transparent latch)
U   14pin  xxxx
U   80pin  SD1812 349 (SMD, without socket)
U   14pin  not installed
U   28pin  KM48C2100J-7 (DRAM, 2Mx8)  ;\
U   20pin  KM44C1000CJ-6 (DRAM, 1Mx4) ; all installed,
U   20pin  KM44C1000CJ-6 (DRAM, 1Mx4) ; together = 4Mx8
U   20pin  KM44C1000CJ-6 (DRAM, 1Mx4) ;
U   20pin  KM44C1000CJ-6 (DRAM, 1Mx4) ;/
X    2pin  16.000MHz
X    2pin  16.257MHz
J   46pin  Cartridge edge (snes)
J   46pin  Cartridge slot (snes)
J   34pin  not installed (alternate floppy connector?)
J   34pin  Floppy data
J    2pin  Floppy supply
BT  2pin   3.6V Battery
```

SNES Cart Copiers - Gamars Copier

Known as:

```
ALMA Super Disk F-16
Gamars Super Disk FC-301
FR-402 Super Disk (bundled with "FR-402 Super 16bit" SNES clone)

2K SRAM at 005000 with REQUIRED mirror at 005800
          3F5Fxx.W  set to FFh,FFh,FFh...
          3F5FC0.R  FDC stat  (bit7,bit5)
          3F5FD2.W  FDC motor? (set to 0Ch,1Ch,08h,0Ch)
          3F5FE4.R  FDC Main Status
          3F5FED.RW FDC Command/Data (emit 03,DF,03)
```

Gamars Puzzle

Aside from the Gamars BIOSes, there's a mis-named ROM-image in the internet: "Gamars (Copier BIOS)", this file is made by the same company, but it's a Puzzle game, not a copier BIOS.

SNES Cart Copiers - Venus (Multi Game Hunter)

MGH (Multi Game Hunter) from Venus.

The 32Kbyte BIOS contains both SNES/65C816 code (entrypoint at [FFFCh]) and Genesis/Z80 code (entrypoint at 0000h).

```
006000..007FFF -- RAM or so
035800..035807 -- I/O Ports
---
006400.R          id "SFCJ"
007D00..007EFF.R  checksummed
035800.W    set to C0h
035801.W    set to A0h
035802.W    set to 0000h or 06h
035803.W    set to 04h
035804.R    disk status?  (bit7,bit6)
035805.W    disk command? (set to 0Bh) (not a uPD765 command?)
035806.W    set to 00h or ([AAh] ROR 1)
035807.W    set to 00h or [ABh]
```

FDC Accress via 80C51 CPU

Like many other copiers, the MGH does use a "normal" MCS3201FN controller, but, it does indirectly access it through a 80C51 CPU. For example,

```
  05       cmd (write sec)                              ;\
  [0B6B]   track          ;\less parameters as than     ; write sector
  [0BA4]   head           ; directly accessing a uPD765 ; command
  [0BA2]   sector         ;/                            ; (at PC=CBAFh)
  [[18]+y] data... (200h bytes)                         ;/
```

SNES Cart Copiers - Others

Component List - Board "GP-003 REV. B" (used in Special Partner)

```
U   14pin  GD74HC04 (hex inverters)
U   16pin  LR74HC158 (decoder/demultiplexer)
U   16pin  LR74HC158 (decoder/demultiplexer)
U   16pin  GD74HC138 (decoder/demultiplexer)
U   16pin  GD74HC138 (decoder/demultiplexer)
U   20pin  PALCE16V8H-25
U   28pin  K-105                                        ;DSP clone?
U   28pin  EPROM (28pin 27C512 64Kx8 installed, optionally 32pin possible)
U   28pin  NEC D43256BGU-70L (uPD43256BGU) (SRAM 32Kx8)
U   28pin  NEC 4364C-20L (SRAM 8Kx8)
U   44pin  GoldStar GM82C765B PL (SMD)
U   44pin  Lattice ispLSI 1016-60LJ B501B06 (SMD)
U    3pin  7805 or so
J2  62pin  to cartridge edge (snes)
J   62pin  cartridge slot (snes game cartridge)
J   62pin  cartridge slot (an expansion slot, not for any game carts)
J   40pin  to DRAM daughterboard
J   34pin  to internal floppy drive
J2   2pin  floppy power supply
J    2pin  external power supply
J6   2pin  jumper (near 34pin floppy cable)
J8   3pin  jumper (near EPROM; maybe ROM size select?)
X    2pin  oscillator (160)
X    2pin  oscillator (?)                               ;DSP clock?
BT   2pin  NiCd 3.6V
```

The board doesn't contain a CIC-clone (unless it's 'hidden' in one of the chips).

Components - Supercom, 24m DSP, CD-ROM, FX-32, High Density, Real Time Save

```
U1  40pin GoldStar GM82C765B (floppy disc controller)
U2  20pin 16V8 (not installed)
U3? 20pin PALCE16V8H-25PC/4
U4  24pin PALCE20V8H-25PC/4
U5  28pin not installed (probably for DSP clone)
U6? 14pin xxx (below U5)
U7? 20pin LS245 (not installed) (8-bit 3-state transceiver)
U8? 24pin PALCE20V8H-25PC/4
U9  20pin 74HC273 (8bit latch with reset)
U10 28pin 27C256G-20 (EPROM 32Kx8) (boots as "FX-32 CD-ROM & DSP, 1994 H.K.")
U11 28pin ST MK4864 (SRAM 8Kx8)
U12 28pin xxx (SRAM ?Kx8)
U13 20pin xxx
U14 20pin xxx
U15 20pin xxx
U16 20pin xxx
U17 20pin xxx
U18 20pin xxx
U19  ?pin Toshiba xxx (16pin chip, mounted in a 20pin socket)
U20 16pin ST101xxx
Q1   3pin 7805
Y1   2pin oscillator
Y2   2pin oscillator (not installed, probably for DSP chip)
J?  34pin floppy data
J?   2pin floppy power
J?   2pin power supply
J3? 25pin DB-25 (parallel port and/or external CD-ROM drive?)
J4  62pin cartridge edge
J5  62pin cartridge slot
J6  40pin to DRAM daughterboard
```

Component List - Double Pro Fighter (CCL) (1994)

```
U1  28pin  Hyundai HY62256ALP-10 (SRAM 32Kx8)
U2  28pin  AM27C512-205DC (EPROM 64Kx8)
U3  N/A    N/A
U4  N/A    N/A
U5  20pin  HD74LS245P (8-bit 3-state transceiver)
U6  20pin  HD74LS245P (8-bit 3-state transceiver)
U7  20pin  HD74LS245P (8-bit 3-state transceiver)
U8  24pin  GoldStar GM76C28A-10 (SRAM 2Kx8)
```

```
U9  16pin  noname-chip-without-part-number (or, marked 10198 on other boards)
U10  3pin  AN7805 (voltage regulator)
U11 14pin  HD74HC00P (quad 2-input NAND gates)
U12 40pin  Goldstar GM82C765B (floppy disc controller)
U13 68pin  Altera EP1810LC-45 D9407
U14 16pin  74HC139 (decoder/demultiplexer)
U15 24pin  PALCE20V8H
U16 20pin  GAL16V8xxx
U17 20pin  PALCxxx
U18 16pin  74HC139 (decoder/demultiplexer)
Y1   2pin  16.00 TDX (16 MHz oscillator)
J1   2pin  power supply input
J2   2pin  power supply connector (alternately to J1 or so, not installed)
P4  50pin  ro dram daughterboard ?
SL1  64pin connector for remove-able snes-or-sega? cartridge edge
SL2  64pin connector for remove-able sega-or-snes? cartridge edge
SL3  62pin cartridge slot (snes)
SL4  64pin cartridge slot (sega genesis)
?     2pin connector for disc drive (supply)
?    34pin connector for disc drive (data)
DRAM Daughterboard
-    40pin connector (to 40pins of the 50pin socket on Double Pro Fighter)
-    20pin NEC 424400-80 (EIGHT pieces)
Optional Parallel Port (plugged into SL3-socket, ie. into SNES slot):
U1   20pin PALCE16V8H-25PC/4
U2   20pin HD74HC245P (8-bit 3-state transceiver) (no latch here ???)
P1   25pin DB-25 parallel port connector
-    62pin cartridge edge (to be plugged into SL3 of Double Pro Fighter)
```

Component List - Super Smart Disc (same as Pro Fighter X?)

```
U   16pin 10198
U   28pin GRAPHIC DSP1-1  (or is it "DCP1-1" or so?)
U   28pin STxxxx (SRAM, ?x8)
U   28pin xxxxxx (SRAM, ?x8)
U   28pin EPROM (28pin chip mounted in 32pin socket)
U   14pin xxxx
U   40pin ICT PA7140T CTM42027JC
U   40pin ICT PA7140T CTM42027JC
U   40pin GoldStar GM82C765B
U   24pin xxxxx (PAL or so)
U    3pin 7805 or so
X    2pin oscillator
P1  64pin cartridge edge (via remove-able adaptor) (snes)
P   62pin cartridge slot (snes)
P   32pin cartridge slot (gameboy)
P   34pin floppy data
P    2pin power supply
P    2pin floppy supply
P   50pin to DRAM daughterboard
```

SNES Cart Copiers - Misc

Parallel Ports (DB-25)

Parallel Ports are used to upload/download data from PCs. Later copiers seem to be additionally using the Parallel Port for connecting CD-ROM drives.
Some copiers have fully working parallel ports installed (eg. Front Fareast), some have them incompletely installed (eg. some Supercom seem to require additional 74LS245 (8-bit 3-state transceiver), and a specially programmed PAL16V8 chip?).
Other copiers don't have any provisions for parallel ports onboard - but can be eventually upgraded externally by plugging a parallel port cartridge into the SNES cartridge slot: There are at least two such upgrade cartridges (one contains pure logic, the other one additionally contains a BIOS upgrade).

DSP Chips

Some copiers include DSP-clones onboard (or do at least have sockets or soldering points for mounting DSP chips), other copiers can be upgraded externally: By plugging a DSP cartridge into the SNES cartridge slot (either a regular game cartridge with DSP chip, or a plain DSP-clone-cartridge without any game in it). Of course, this will work only with the correct DSP chip (DSP1 for most games; unless there are any DSP clones that support more than one DSP chip at one?). Another problem may be I/O addresses (different games expect DSP chips at different addresses).

Batteries

Some boards contain batteries for the internal SRAM. Either 3V Lithium cells (coin-shaped), or rechargeable 3.6V NiCd batteries (usually with blue coating, which tend to leak acid, and to destroy wires on the PCB). Other boards don't have any batteries at all (they are said to use capacitors instead of batteries, which might be nonsense, or might last only a few minutes?) (there seems to be no way to switch-off the external power-supply, so batteries aren't needed to power SRAM) (eventually some boards might even power the DRAM in standby mode (?), which would require a DRAM refresh generator). And, aside from battery-backup, most or all copiers are allowing to save SRAM to floppy.

SNES Cart Copiers - Floppy Disc Controllers

FDC Chips

Most (or all) SNES copiers are using one of the following FDCs:

```
40pin  GM82C765B (DIP)  (Supercom, Ufo, Pro Fighter, Smart Disc, Bung?)
44pin  GM82C765B (SMD)  (Wild Card, GP-003)
68pin  MCS3201FN (SMD)  (used by OLD copiers: Super Magic Drive)
68pin  MCCS3201FN (SMD) (used by OLD copiers: Supercom & Super Magicom)
```

FDC Address Decoding

The 68pin MCS3201FN chips include a 10bit address bus (for decoding address 3F0h-3F7h on IBM PCs; whereas, SNES copiers are using only the lower some address bits), and an 8bit General Purpose Input. The 40pin/44pin GM82C765B chips include a 1bit address bus bundled with 3 select lines, and have no General Purpose Input register.

```
GM82C765B   MCS3201FN  Dir  Register
N/A         A0-A2=0    R    General Purpose Input (pins I0..I7)
/LDOR       A0-A2=2    W    Motor Control (bit0-7)
/CS+A0=0    A0-A2=4    R    Main Status  (NEC uPD765 compatible)
/CS+A0=1    A0-A2=5    RW   Command/Data (NEC uPD765 compatible)
/LDCR       A0-A2=7    W    Transfer Rate (Density) (bit0-1)
N/A         A0-A2=7    R    Bit7=DiskChange, Bit6-0=Zero
```

Accordingly, MCS3201FN ports are always ordered as shown above, whilst GM82C765B ports can be arranged differently (in case of the Super Wild Card, Front Fareast kept them arranged the same way as on their older Super Magicom).

FDC Command/Data and Main Status
SNES Cart Copiers - Floppy Disc NEC uPD765 Commands

FDC Motor Control
```
GM Bit0-7:  DSEL ,X    ,/RES,DMAEN,MOTOR1,MOTOR2,X     ,MSEL
MCS Bit0-7: DSEL0,DSEL1,/RES,DMAEN,MOTOR1,MOTOR2,MOTOR3,MOTOR4
```
Note that, for whatever reason, most SNES Copiers are using the SECOND drive (ie. DSEL=1 instead of DSEL=0, and MOTOR2=1 instead of MOTOR1=1).

Transfer Rate (Density)
```
Val  Usage                 MCS3201FN       GM82C765B
00h  HD (high density)     500K if /RWC=1  MFM:500K or FM:250K
01h  DD 5.25" (double den) 300K if /RWC=0  MFM:300K if DRV=1, 250K if DRV=0
02h  DD 3.5"(double den)   250K if /RWC=0  MFM:250K or FM:125K
03h  N/A                   Reserved        125K
```

Disk Change (MCS3201FN only) (not GM82C765B)
```
7   Disk Change Flag
6-0 Unused (zero)
```
Possibly useful, but purpose/usage is unclear. According to the datasheet it is for "diagnostics" purposes. Unknown when the flag gets reset, and unknown for which drive(s) it does apply.

SNES Cart Copiers - Floppy Disc NEC uPD765 Commands

Accessing the FDC 765
The Data Register is used to write Commands and Parameters, to read/write data bytes, and to receive result bytes. These three operations are called Command-, Execution-, and Result-Phase. The Main Status Register signalizes when the FDC is ready to send/receive the next byte through the Data Register.

Command Phase
A command consists of a command byte (eventually including the MF, MK, SK bits), and up to eight parameter bytes.

Execution Phase
During this phase, the actual data is transferred (if any). Usually that are the data bytes for the read/written sector(s), except for the Format Track Command, in that case four bytes for each sector are transferred.

Result Phase
Returns up to seven result bytes (depending on the command) that are containing status information. The Recalibrate and Seek Track commands do not return result bytes directly, instead the program must wait until the Main Status Register signalizes that the command has been completed, and then it must (!) send a Sense Interrupt State command to 'terminate' the Seek/Recalibrate command.

FDC Command Table
```
Command     Parameters              Exm Result              Description
02+MF+SK    HU TR HD ?? SZ NM GP SL <R> S0 S1 S2 TR HD NM SZ read track
03          XX YY                    -                      specify spd/dma
04          HU                       -  S3                  sense drive state
05+MT+MF    HU TR HD SC SZ LS GP SL <W> S0 S1 S2 TR HD LS SZ write sector(s)
06+MT+MF+SK HU TR HD SC SZ LS GP SL <R> S0 S1 S2 TR HD LS SZ read sector(s)
07          HU                       -                      recalib.seek TP=0
08          -                        -  S0 TP               sense int.state
09+MT+MF    HU TR HD SC SZ LS GP SL <W> S0 S1 S2 TR HD LS SZ wr deleted sec(s)
0A+MF       HU                       -  S0 S1 S2 TR HD LS SZ read ID
0C+MT+MF+SK HU TR HD SC SZ LS GP SL <R> S0 S1 S2 TR HD LS SZ rd deleted sec(s)
0D+MF       HU SZ NM GP FB          <W> S0 S1 S2 TR HD LS SZ format track
0F          HU TP                    -                      seek track n
11+MT+MF+SK HU TR HD SC SZ LS GP SL <W> S0 S1 S2 TR HD LS SZ scan equal
19+MT+MF+SK HU TR HD SC SZ LS GP SL <W> S0 S1 S2 TR HD LS SZ scan low or equal
1D+MT+MF+SK HU TR HD SC SZ LS GP SL <W> S0 S1 S2 TR HD LS SZ scan high or eq.
```

Parameter bits that can be specified in some Command Bytes are:
```
MT  Bit7  Multi Track (continue multi-sector-function on other head)
MF  Bit6  MFM-Mode-Bit (Default 1=Double Density)
SK  Bit5  Skip-Bit (set if secs with deleted DAM shall be skipped)
```

Parameter/Result bytes are:
```
HU  b0,1=Unit/Drive Number, b2=Physical Head Number, other bits zero
TP  Physical Track Number
TR  Track-ID (usually same value as TP)
HD  Head-ID
SC  First Sector-ID (sector you want to read)
SZ  Sector Size (80h shl n) (default=02h for 200h bytes)
LS  Last Sector-ID (should be same as SC when reading a single sector)
GP  Gap (default=2Ah except command 0D: default=52h)
SL  Sectorlen if SZ=0 (default=FFh)
Sn  Status Register 0..3
FB  Fillbyte (for the sector data areas) (default=E5h)
NM  Number of Sectors (default=09h)
XX  b0..3=headunload n*32ms (8" only), b4..7=steprate (16-n)*2ms
YY  b0=DMA_disable, b1-7=headload n*4ms (8" only)
```

Format Track: output TR,HD,SC,SZ for each sector during execution phase
Read Track: reads NM sectors (starting with first sec past index hole)
Read ID: read ID bytes for current sec, repeated/undelayed read lists all IDs
Recalib: walks up to 77 tracks, 80tr-drives may need second recalib if failed
Seek/Recalib: All read/write commands will be disabled until succesful senseint
Senseint: Set's IC if unsuccesful (no int has occured) (until IC=0)

FDC Status Registers
The Main Status register can be always read through an I/O Port. The other four Status Registers cannot be read directly, instead they are returned through the data register as result bytes in response to specific commands.

Main Status Register (I/O Port)
```
b0..3  DB  FDD0..3 Busy (seek/recalib active, until succesful sense intstat)
b4     CB  FDC Busy (still in command-, execution- or result-phase)
b5     EXM Execution Mode (still in execution-phase, non_DMA_only)
b6     DIO Data Input/Output (0=CPU->FDC, 1=FDC->CPU) (see b7)
b7     RQM Request For Master (1=ready for next byte) (see b6 for direction)
```
Status Register 0
```
b0,1   US  Unit Select (driveno during interrupt)
b2     HD  Head Address (head during interrupt)
b3     NR  Not Ready (drive not ready or non-existing 2nd head selected)
b4     EC  Equipment Check (drive failure or recalibrate failed (retry))
b5     SE  Seek End (Set if seek-command completed)
b6,7   IC  Interrupt Code (0=OK, 1=aborted:readfail/OK if EN, 2=unknown cmd
           or senseint with no int occured, 3=aborted:disc removed etc.)
```
Status Register 1
```
b0     MA  Missing Address Mark (Sector_ID or DAM not found)
b1     NW  Not Writeable (tried to write/format disc with wprot_tab=on)
b2     ND  No Data (Sector_ID not found, CRC fail in ID_field)
b3,6   0   Not used
b4     OR  Over Run (CPU too slow in execution-phase (ca. 26us/Byte))
b5     DE  Data Error (CRC-fail in ID- or Data-Field)
b7     EN  End of Track (set past most read/write commands) (see IC)
```
Status Register 2
```
b0     MD  Missing Address Mark in Data Field (DAM not found)
b1     BC  Bad Cylinder (read/programmed track-ID different and read-ID = FF)
b2     SN  Scan Not Satisfied (no fitting sector found)
b3     SH  Scan Equal Hit (equal)
b4     WC  Wrong Cylinder (read/programmed track-ID different) (see b1)
b5     DD  Data Error in Data Field (CRC-fail in data-field)
b6     CM  Control Mark (read/scan command found sector with deleted DAM)
b7     0   Not Used
```
Status Register 3
```
b0,1   US  Unit Select (pin 28,29 of FDC)
b2     HD  Head Address (pin 27 of FDC)
b3     TS  Two Side (0=yes, 1=no (!))   GM82C765: Also WP (same as bit6)?
b4     T0  Track 0 (on track 0 we are)
b5     RY  Ready (drive ready signal)   GM82C765: Always 1=Ready
b6     WP  Write Protected (write protected)
b7     FT  Fault (if supported: 1=Drive failure) GM82C765: Always 0=Okay
```

Notes:
Before accessing a disk you should first Recalibrate the drive, that'll move the head backwards until it reaches Track 0 (that's required to initialize the FDCs track counter). On a 80 track drive you may need to repeat that in case that the first recalibration attempt wasn't successful (that's because the FDC stops searching after 77 steps) (at least older uPD765 chips did so, maybe the MCS3201FN/GM82C765B chips don't).

Now if you want to format, read or write a sector on a specific track you must first Seek that track (command 0Fh). That'll move the read/write head to the physical track number. If you don't do that, then the FDC will attempt to read/write data to/from the current physical track, independendly of the specified logical Track-ID.

The Track-, Sector-, and Head-IDs are logical IDs only. These logical IDs are defined when formatting the disk, and aren't required to be identical to the physical Track, Sector, or Head numbers. However, when reading or writing a sector you must specify the same IDs that have been used during formatting.

Despite of the confusing name, a sector with a "Deleted Data Address Mark" (DAM) is not deleted. The DAM-flag is just another ID-bit, and (if that ID-bit is specified correctly in the command) it can be read/written like normal data sectors.

DMA/IRQ
Most (or all) SNES copiers don't support DMA or IRQs (some are allowing to poll the IRQ flag by software I/O).

Terminal Count (TC)
*** Below info applies to Amstrad CPC with uPD765 chip.
*** Unknown if anything similar applies to SNES with MCS3201FN/GM82C765B chips.
At the end of a successful read/write command, the program should send a Terminal Count (TC) signal to the FDC. However, in the CPC the TC pin isn't connected to the I/O bus, making it impossible for the program to confirm a correct operation. For that reason, the FDC will assume that the command has failed, and it'll return both Bit 6 in Status Register 0 and Bit 7 in Status Register 1 set. The program should ignore this errormessage.

SNES Cart Copiers - Floppy Disc FAT12 Format

The SNES Copier floppy format is compatible to that used under DOS on PCs.
Typical formats are 3.5", Double Density, 80 Tracks/9 Sectors, Double Sided (720KB). The Sectors are logically numbered 01h..09h, and each sized 200h bytes.
XXX HD-disks have more sectors
XXX snes copiers are usually HD (or maybe some are DD?)
XXX snes copiers support 1.44MB and 1.6MB (FDFORMAT-like)

Boot-Record
The first sector is always used as bootsector, giving information about the usage of the following sectors, and including the boot procedure (for loading MSDOS etc).
```
00-02      80x86 boot procedure (jmp opcode) (not used for SNES)
03-0A      ascii disk name
0B-0C      bytes / sector
0D         sectors / cluster
0E-0F      sectors / boot-record
10         number of FAT-copys
11-12      entrys / root-directory
13-14      sectors / disk
15         ID: F8=hdd, F9=3.5", FC=SS/9sec, FD=DS9, FE=SS8,FF=DS8
16-17      sectors / FAT
18-19      sectors / track
1A-1B      heads / disk
1C-1D      number of reserved sectors
1E-1FF     MSX boot procedure (Z80 code) (not used for SNES)
```

FAT and FAT copy(s)
The following sectors are occupied by the File Allocation Table (FAT), which contains 12- or 16-bit entries for each cluster:

```
(0)000      unused, free
(0)001      ???
(0)002...   pointer to next cluster in chain (0)002..(F)FEF
(F)FF0-6    reserved (no part of chain, not free)
(F)FF7      defect cluster, don't use
(F)FF8-F    last cluster of chain
```

Number and size of FATs can be calculated by the information in the boot sector.

Root directory
The following sectors are the Root directory, again, size depends on the info in bootsector. Each entry consists of 32 bytes:

```
00-07       Filename (first byte: 00=free entry,2E=dir, E5=deleted entry)
08-0A       Filename extension
0B          Fileattribute
0C-15       reserved
16-17       Timestamp: HHHHHMMM, MMMSSSSS
18-19       Datestamp: YYYYYYYM, MMMDDDDD
1A-1B       Pointer to first cluster of file
1C-1F       Filesize in bytes
```

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.

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 (0)002, followed by number (0)003, (0)004, and so on.

Special Features
Unknown if any copiers support sub-directories.
Unknown if any copiers support long file names.
Unknown if any copiers support compressed files (ZIP or such).

SNES Cart Copiers - BIOSes

Copier BIOSes

Name			I/O	BIOS Size		
Double Pro Fighter (1994)			2800	64K(6+6+16)		
Gamars Puzzle (not a Copier BIOS)			-	1M (32x32K)	GAMARS~5	1,048,576
Gamars Super Disk FC-301 V6.0	Kaiser94	5Fxx		64K (1x64K)	GAMARS~4	65,536
Gamars Super Disk FC-301 V7.13	Kaiser94	5Fxx	256K	(4x64K)	GAMARS~3	262,144
Gamars Super Disk FC-301 V7.16	Kaiser94	5Fxx	256K	(4x64K)	GAMARS~2	262,144
Game Doctor SF 3 V3.3C			8000	32K (1x32K)	GAMEDO~3	32,768
Game Doctor SF 6 V6.2 (Professor SF)			8000	64K (2x32K)	GAMEDO~4	65,536
Game Doctor SF 6 V6.21 (Professor SF)			8000	64K (2x32K)	GAMEDO~6	65,536
Game Doctor SF 7 V7.11 (Professor SF 2)			8000	64K (2x32K)	GAMEDO~2	65,536
Multi Game Hunter V1.2 (Venus)			5800	32K (1x32K)	MULTIG~2	32,768
Multi Game Hunter V1.3 (Venus)			5800	32K (1x32K)	MULTIG~3	32,768
Multi Game Hunter V1.4 (Venus)			5800	32K (1x32K)	MULTIG~3	32,768
Pro Fighter Q (H.K.)		xx-xx-93	2800	16K (1x16K)	SUPERP~1	16,xxx
Supercom Partner A [o1]			2800	16K (1x16K)	SUPERC~1	3,145,728
Supercom Pro 2 (CCL) ports=FFE		06-21-92	FFE	8K (1x8K)	SUPERC~2	32,768
Super Disk Interceptor v5.2 (Sane Ting)			A000	32K (4x8K)	SUPERD~1	32,768
Super Magicom V1H (Front/CCL)		12-23-91	FFE	8K (1x8K)	SUPERM~2	32,768
Super Magicom V31 (Front/CCL)		xx-xx-92	FFE	8K (1x8K)	SUPERM~4	32,768
Super Magicom V3H SoftUpgrade		xx-xx-9x	FFE	32K (DRAM)	SUPERM~8	32,768
Super Pro Fighter (H.K.)		xx-xx-93	2800	16K (1x16K)	SUPERP~1	16,xxx
Super Pro Fighter (H.K.) [a1]		xx-xx-93	2800	16K (1x16K)	SUPERP~1	16,xxx
Super Wild Card V1.6	(Front)	93-01-26	FFE	16K (2x8K)	SUPERW~6	16,384
Super Wild Card V1.8	(Front)	93-02-19	FFE	16K (2x8K)	SUPERW~7	16,384
Super Wild Card V2.0XL	(Front)	93-04-12	FFE	16K (2x8K)	SUPERW~9	16,384
Super Wild Card V2.1B	(Front)	93-04-28	FFE	16K (2x8K)	SUPER~10	16,384
Super Wild Card V2.1C	(Front)	93-04-28	FFE	16K (2x8K)	SUPER~11	16,384
Super Wild Card V2.2CC	(Front)	93-05-03	FFE	16K (2x8K)	SUPER~12	16,384
Super Wild Card V2.6CC	(Front)	93-07-17	FFE	16K (2x8K)	SUPER~15	16,384
Super Wild Card V2.6F	(Front)	93-07-17	FFE	16K (2x8K)	SUPER~16	16,384
Super Wild Card V2.6FX	(Front)	93-07-17	FFE	16K (2x8K)	SUPER~17	16,384
Super Wild Card V2.7CC	(Front)	93-12-07	FFE	16K (2x8K)	SUPER~18	16,384
Super Wild Card V2.8CC	(Front)	06-08-94	FFE	16K (2x8K)	SUPER~19	16,384
Super Wild Card V2.8CC [o1]		06-28-94	FFE	16K (2x8K)	SUPER~22	65,536
Super Wild Card DX		10-14-94	FFE	256K (32x8K)	SUPERW~2	262,144
Super Wild Card DX		11-03-94	FFE	256K (32x8K)	SUPERW~3	262,144
Super Wild Card DX96		01-04-96	FFE	256K (32x8K)	SUPERW~1	262,144
Super Wild Card DX2		06-08-96	FFE	256K (32x8K)	SUPERW~4	262,144
UFO - Super Drive PRO 3 [o1 as 4x8K]			FFE	8K (1x8K)	UFOSUP~1	32,768
UFO - Pro 6			?	? (?)		
UFO - Super UFO Pro-7 V7.3		1994	2184	64K (2x32K)	SUPERU~1	65,536
UFO - Super UFO Pro-8 V8.1		1995	2184	128K (4x32K)	SUPERU~2	131,072
UFO - Super UFO Pro-8 V8.8c		1995	2184	256K (8x32K)	SUPERU~3	262,144

SNES Hotel Boxes and Arcade Machines

Nintendo Super System (NSS) (USA) (1991)
Arcade Cabinet. Contains up to three special cartridges (with one game each).
NSS Memory and I/O Maps
NSS I/O Ports - Control Registers
NSS I/O Ports - Button Inputs and Coin Control
NSS I/O Ports - RTC and OSD
NSS I/O Ports - EEPROM and PROM
NSS BIOS and INST ROM Maps

Super Famicom Box (Japan/Nintendo/HAL) (1993)
For use in hotel rooms. Typically contains two special multi-carts (which, together, contain a menu-program and 5 games).

DS-109S (Third-Party/Japan?) (Osaka Tu...?)
For hotels or so. Contains 10 regular SNES/SFC cartridges. Game selection is done via a push button and single-digit 7-segment LED display. Not much more known about the hardware.

Nintendo Gateway System / Lodgenet (USA/Third-Party)
For hotels and airports or so. Reportedly offers 18 SNES games or so (later versions allow games for other/newer consoles). There seem to be no photos of the hardware. With the "net" in in "Lodgenet" mind... it might be "server based" (with only Controller & TV-Set located in the hotel room, and the actual Console & Games located elsewhere?).

NSS Memory and I/O Maps

Z80 Memory Map
```
0000h-7FFFh : 32K BIOS
8000h-9FFFh : 8K RAM (upper 4K with write-protect)
A000h       : EEPROM Input (R)
C000h-DFFFh : Upper 8K of 32K Instruction EPROM (in Cartridge) (INST-ROM)
E000h       : EEPROM Output (W)
Exxxh       : PROM Input AND Output AND Program Code (RST opcodes) (R/W/EXEC)
```
Note: For some reason, Nintendo has stored the 8K INST-ROM in 32K EPROMs - the first 24K of that EPROMs are unused (usually 00h-filled or FFh-filled, and EPROM pins A13 and A14 are wired to VCC, so there is no way to access the unused 24K area).

Z80 IN-Ports
```
Port 00h.R - Joypad Buttons and Vsync Flag
Port 01h.R - Front-Panel Buttons & Game Over Flag
Port 02h.R - Coin and Service Buttons Inputs
Port 03h.R - Real-Time Clock (RTC) Input
Port 04h-07h.R - Unused (returns FFh)
```
Port 0008h..FFFFh are mirrors above ports (whereof, mirrors at xx00h..xx03h are often used).

Z80 OUT-Ports
```
Port 00h/80h.W - NMI Control and RAM-Protect
Port 01h/81h.W - Unknown and Slot Select
Port 02h/82h/72h/EAh.W - Real-Time Clock (RTC) and On-Screen Display (OSD)
Port 03h/83h.W - Unknown and LED control
Port 84h.W - Coin Counter Outputs
Port 05h.W - Unknown (bug?)
Port 06h.W - Unused
Port 07h.W - SNES Watchdog: Acknowledge SNES Joypad Read Flag
```
These ports seem to be decoded by A0..A2 only (upper address bits are sometimes set to this or that value, but seem to have no meaning).

SNES Memory Map
Normal SNES memory map, plus some special registers:
```
4100h/Read.Bit0-7  - DIP-Switches (contained in some NSS cartridges)
4016h/Write.Bit0   - Joypad Strobe (probably clears the SNES Watchdog flag?)
                       (OR, maybe that occurs not on 4016h-writes,
                       but rather on 4016h/4017h-reads, OR elsewhere?)
4016h/Write.Bit2   - Joypad OUT2 indicates Game Over (in Skill Mode games)
4016h/4017h/4218h..421Bh - Joypad Inputs (can be disabled)
```

NSS I/O Ports - Control Registers

Port WHERE.W
Somewhere, following OUTPUT signals should be found:
```
SNES Reset Signal (maybe separate CPU/PPU resets, and stop, as on PC10)
SNES Joypad Disable?
Maybe support for sending data from Z80 to SNES (eg. to 4016h/4017h/4213h)?
```

Port 00h/80h.W - NMI Control and RAM-Protect
```
  7-4 Unknown/unused      (should be always 0)
  3     Maybe SNES CPU/PPU reset (usually same as Port 01h.W.Bit1)
  2   RAM at 9000h-9FFFh  (0=Disable/Protect, 1=Enable/Unlock)
  1   Looks like maybe somehow NMI Related ?    ;\or one of these is PC10-style
  0   Looks like NMI Enable                     ;/hardware-watchdog reload?
```
Usually accessed as "Port 80h", sometimes as "Port 00h".

Port 01h/81h.W - Unknown and Slot Select
```
  7     Maybe SNES Joypad Enable? (0=Disable/Demo, 1=Enable/Game)
  6   Unknown/unused        (should be always 0)
  5   SNES Sound Mute       (0=Normal, 1=Mute) (for optional mute in demo mode)
  4   Unknown  ;from INST-ROM flag! (Lo/HiROM, 2-player, zapper, volume or so?)
  3-2 Slot Select (0..2=1st..3rd Slot, 3=None) (mapping to both SNES and Z80)
  1     Maybe SNES CPU pause?  (cleared on deposit coin to continue) (1=Run)
  0     Maybe SNES CPU/PPU reset?   (0=Reset, 1=Run)
```
Sometimes accessed as "Port 81h", sometimes as "Port 01h".

Port 03h/83h.W - Unknown and LED control
```
  7     Layer SNES Enable?             (used by token proc, see 7A46h) SNES?
  6     Layer OSD Enable?
  5-4 Unknown/unused (should be always 0)
  3   LED Instructions (0=Off, 1=On)  ;-glows in demo (prompt for INST button)
  2   LED Game 3       (0=Off, 1=On)  ;\
  1   LED Game 2       (0=Off, 1=On)  ; blinked when enough credits inserted
  0   LED Game 1       (0=Off, 1=On)  ;/
```
Usually accessed as "Port 83h", sometimes as "Port 03h".

Port 05h.W - Unknown (bug?)
```
  7-0 Unknown
```
Accessed only as "Port 05h" (via "outd" opcode executed 5 times; but that seems to be just a bugged attempt to access Port 04h downto 00h).

Port 07h.W - SNES Watchdog: Acknowledge SNES Joypad Read Flag
```
  7-0 Unknown/unused (write any dummy value)
```
Accessed only as "Port 07h". Writing any value seems to switch Port 00h.R.Bit7 back to "1". That bit is used for the SNES Watchdog feature; the SNES must read joypads at least once every some frames (the exact limit can be set in INST ROM).
If the watchdog expires more than once, then the game is removed from the cartridge list, and used credits are returned to the user (then allowing to play other games; as long as there are any other games installed).
Note: Judging from hardware tests, there seem to be other ways to acknowledge the flag (probably via Port 07h.R, or maybe even via Port 00h.R itself).

NMI
The NMI source is unknown. Maybe Vblank/Vsync, maybe from SNES or OSD, or some other timer signal.

Game/Demo-Mode Detection
The original NSS games seem to be unable to detect if a coin is inserted (ie. if they should enter game or demo mode). However, it's possible to do that kind of detection:
Joypad Disable does much like disconnecting the joypad, so one can check the 17th joypad bit to check if the joypad is connected/enabled (aka if money is inserted). The Magic Floor game is using that trick to switch between game and demo mode (this has been tested by DogP and works on real hardware, ie. the NSS does really disable the whole joypad bitstream, unlike the PC10 which seems to disable only certain buttons).

NSS I/O Ports - Button Inputs and Coin Control

Port 00h.R - Joypad Buttons
```
  7   SNES Watchdog (0=SNES did read Joypads, 1=Didn't do so) (ack via 07h.W)
  6   Vsync from OSD or SNES (?)  (0=Vsync, 1=No) (zero for ca. 3 scanlines)
  5   Button "Joypad Button B?"   (0=Released, 1=Pressed)
  4   Button "Joypad Button A"    (0=Released, 1=Pressed)
  3   Button "Joypad Down"        (0=Released, 1=Pressed)
  2   Button "Joypad Up"          (0=Released, 1=Pressed)
  1   Button "Joypad Left"        (0=Released, 1=Pressed)
  0   Button "Joypad Right"       (0=Released, 1=Pressed)
```

Port 01h.R - Front-Panel Buttons & Game Over Flag
```
  7   From SNES Port 4016h.W.Bit2 (0=Game Over Flag, 1=Normal) (Inverted!)
  6   Button "Restart"            (0=Released, 1=Pressed) ;-also resets SNES?
  5   Button "Page Up"            (0=Released, 1=Pressed)
  4   Button "Page Down"          (0=Released, 1=Pressed)
  3   Button "Instructions"       (0=Released, 1=Pressed)
  2   Button "Game 3"             (0=Released, 1=Pressed) ;\if present (single
  1   Button "Game 2"             (0=Released, 1=Pressed) ; cartridge mode does
  0   Button "Game 1"             (0=Released, 1=Pressed) ;/without them)
```

Port 02h.R - Coin and Service Buttons Inputs
```
  7-3 Unknown/unused (seems to be always 0)
  2   Service Button (1=Pressed: Add Credit; with INST button: Config)
  1   Coin Input 2   (1=Coin inserted in coin-slot 2)
  0   Coin Input 1   (1=Coin inserted in coin-slot 1)
```

Port 84h.W - Coin Counter Outputs
```
  7-2 Unknown/unused (should be always 0)
  1   Coin Counter 2 (0=No change, 1=Increment external counter)
  0   Coin Counter 1 (0=No change, 1=Increment external counter)
```
Accessed only as "Port 84h". To increase a counter, the bit should be set for around 4 frames, and cleared for at least 3 frames (before sending a second pulse).

NSS I/O Ports - RTC and OSD

Port 03h.R - Real-Time Clock (RTC) Input
```
  7-1 Unknown/unused    (seems to be always 7Eh, ie. all seven bits set)
  0   RTC Data In       (0=Low=Zero, 1=High=One)
```

Port 02h/82h/72h/EAh.W - Real-Time Clock (RTC) and On-Screen Display (OSD)
```
7   OSD Clock ?       (usually same as Bit6)  ;\Chip Select when Bit6=Bit7 ?
6   OSD Clock ?       (usually same as Bit7)  ;/
5   OSD Data Out      (0=Low=Zero, 1=High=One)
4   OSD Special       (?)  ... or just /CS ? (or software index DC3F/DD3F?)
3   RTC /CLK          (0=Low=Clock,  1=High=Idle)             ;S-3520
2   RTC Data Out      (0=Low=Zero,   1=High=One)
1   RTC Direction     (0=Low=Write,  1=High=Read)
0   RTC /CS           (0=Low/Select, 1=High/No)
```
RTC is accessed via "Port 82h", OSD via "Port 02h/72h/EAh". For OSD access, the BIOS toggles a LOT of data (and address) lines; not quite clear which of those lines are OSD CLK and OSD Chip Select.

Real-Time Clock (S-3520)
The NSS-BIOS supports year 1900..2099 (century 00h=19xx, FFh=20xx is stored in RAM at 8F8Dh/978Dh/9F8Dh; in the two version "03" BIOSes). The current time is shown when pressing Restart in the Bookkeeping screen.
RTC S-3520 (Real-Time Clock)

OSD On-Screen Display (M50458-001SP)
NSS On-Screen Controller (OSD)

NSS I/O Ports - EEPROM and PROM

Memory A000h.R - EEPROM Input
```
7   EEPROM Data In (0=Low=Zero, 1=High=One)
6   EEPROM Ready   (0=Low=Busy, 1=High=Ready)
5-0 Unknown/unused
```

Memory E000h.W - EEPROM Output
```
7   Unknown/set     (should be always 1)
6-5 Unknown/unused  (should be always 0)
4   EEPROM Clock    (0=Low=Clock, 1=High=Idle) ;(Data In/Out must be stable
3   EEPROM Data Out (0=Low=Zero, 1=High=One)   ;on raising CLK edge)
2-1 Unknown/unused  (should be always 0)       ;(and updated on falling edge)
0   EEPROM Select   (0=High=No, 1=Low=Select)
```

Note
E000h (W) and Exxxh (W) are probably mirrors of each other. If so, some care should be taken not to conflict PROM and EEPROM accesses.

Memory Exxxh.R.W.EXEC - Ricoh RP5H01 serial 72bit PROM (Decryption Key)
Data Write:
```
7-5  Unknown/unused
4    PROM Test Mode (0=Low=6bit Address, 1=High=7bit Address)
3    PROM Clock     (0=Low, 1=High) ;increment address on 1-to-0 transition
2-1  Unknown/unused
0    PROM Address Reset (0=High=Reset Address to zero, 1=Low=No Change)
```
Data Read and Opcode Fetch:
```
7-5  Always set (MSBs of RST Opcode)
4    PROM Counter Out (0=High=One, 1=Low=Zero) ;PROM Address Bit5
3    PROM Data Out    (0=High=One, 1=Low=Zero)
2-0  Always set (LSBs of RST Opcode)
```
The BIOS accesses the PROM in two places:
```
1st PROM check: Accessed via E37Fh, this part decrypts the 32h-byte area.
  the first data bit is read at a time when PROM reset is still high,
  and reset is then released after reading that data bit. At this point,
  there's a critical glitch: If the data bit was 1=Low, then the decryption
  code chooses to issue a 1-to-0 CLK transition at SAME time as when
  releasing reset - the PROM must ignore this CLK edge (otherwise half
  of the games won't work).
2nd PROM check: Accessed via EB27h, this part decrypts the double-encrypted
  title (from within the 32h-byte area) and displays on the OSD layer,
  alongsides it does verify a checksum at DC3Fh.
  Note: The program code hides in the OSD write string function, and gets
  executed when passing invalid VRAM addresses to it; this is usually done
  via Token 06h.
  This is initially done shortly after the 1st PROM check (at that point
  just for testing DC3Fh, with "invisible" black-on-black color attributes).
```
And, there are two more (unused/bugged) places:
```
3rd PROM check: Accessed via FB37h, this part is similar to 2nd PROM check,
  but sends garbage to OSD screen, and is just meant to verify checksum at
  DD3Fh. However, this part seems to be bugged (passing FB37h to the RST
  handler will hang the BIOS). The stuff would be invoked via Token 4Eh,
  but (fortunately) the BIOS is never doing that.
4th PROM check: Accessed via ExExh, this part is comparing the 1st eight
  bytes of the PROM with a slightly encrypted copy in INST ROM. However,
  in F-Zero, the required pointer at [2Eh-2Fh] in the 32h-byte area is
  misaligned, thus causing the check to fail. The stuff would be invoked
  from inside of NMI handler (when [80ECh] nonzero), but (fortunately) the
  BIOS is never doing that.
```
Note: All (used) PROM reading functions use RST vectors which are executing Z80 code in INST ROM. Accordingly, the code in INST ROM can be programmed so that it works with PROM-less cartridges.

PROM Dumps
Theoretically, dumping serial PROMs is ways easier than dumping parallel ROMs/EPROMs - but, as by now, nobody does ever seem to have done this. Anyways, with a brute-force program, it's possible to find matching PROM values for decrypting known title strings.
```
Title                 PROM content
ActRaiser             B9,4B,F5,72,E4,9E,25,FF,F2,F2,00,00,F2,F2,00,00
AMAZING TENNIS        2D,EB,21,3B,9A,81,86,93,57,57,00,00,57,57,00,00
F-ZERO                49,63,FA,03,B5,DF,F6,17,B7,B7,00,00,B7,B7,00,00
LETHAL WEAPON         7F,9B,42,99,D4,C2,A9,0A,CB,CB,00,00,CB,CB,00,00
NCAA Basketball       DB,35,54,07,A0,EF,A2,72,F8,F8,00,00,F8,F8,00,00
New Game 1 [Contra 3] 3A,BC,E6,47,10,DD,45,AF,FC,FC,00,00,FC,FC,00,00
ROBOCOP 3             6A,06,DC,99,5F,3A,5C,D1,5D,5D,00,00,5D,5D,00,00
Super Mario World     AE,D4,A8,1C,EC,DA,8D,EA,7D,7D,00,00,7D,7D,00,00
```

```
SUPER SOCCER          6C,57,7E,3C,8F,1F,AB,F2,3D,3D,00,00,3D,3D,00,00
Super Tennis          86,B7,8E,BD,74,A3,6E,56,9F,9F,00,00,9F,9F,00,00
The Addams Family     C1,70,F2,7F,3A,EC,D3,02,67,67,00,00,67,67,00,00
The Irem Skins Game   D7,3F,FE,6A,B7,3A,18,AA,D6,D6,00,00,D6,D6,00,00
```

Mitsubishi M6M80011 64x16 Serial EEPROM Protocol

All vallues transferred LSB first.

```
Write Enable:  Send C5h,xxh
Write Disable: Send 05h,xxh
Write Word:    Send 25h,addr, Send lsb,msb
Read Word:     Send 15h,addr, Read lsb,msb
Read Status:   Send 95h,mode, Read stat...
  (mode: 0=Busy, 1=WriteEnable, 2=ECC Flag)
  (stat: endless repeated bits, 0=Busy/WriteEnable/ECC_Correct)
  (                             1=Ready/WriteDisable/ECC_Incorrect)
```

M6M80011 Pin-Out (2x4pin version)

```
1=/CS, 2=/CLK, 3=DTA.IN, 4=DTA.OUT, 5=GND, 6=RESET, 7=RDY/BUSY, 8=VCC
```

NSS EEPROM Format (Coinage Settings)

```
00h-3Bh Fifteen 4-byte chunks (unused entry when 1st byte = 00h)
         Byte0: Upper Nibble: Checksum (all other 7 nibbles added together)
         Byte0: Lower Nibble: Price (Number of credits for this game, 1..9)
         Byte1: GameID
         Byte2: Time Minutes (BCD) (time limit per game)
         Byte3: Time Seconds (BCD) (time limit per game)
3Ch     Right Coinage and Unused (bit7-4, unused, but must be 1..9)
3Dh     Left Coinage and Flags (bit7=Music, bit6=Freeplay, bit5-4=unused)
3Eh-3Fh Checksum (all bytes at [00h..3Dh] added together)
40h-7Fh Backup Copy of 00h..3Fh
```

NSS BIOS and INST ROM Maps

NSS BIOS ROM (32K mapped to 0000h-7FFFh)

```
0000h   Reset Vector
0008h   RST Handlers (internally used by PROM checks)
0066h   NMI Handler (unknown source, probably Vblank or Vsync or so)
3FFDh   Hardcoded Token Address (used by F-Zero INST ROM)
5F30h   Hardcoded Return-Address from 2nd PROM check in INST ROM
```

NSS INST ROM (8K mapped to C000h-DFFFh)

```
[C034h]+00h..31h   Encrypted Data (to be decrypted via PROM data)
[C034h]+32h..33h   Chksum on above 32h bytes (all BYTEs added together)
[C67Fh]+C600h      RST 38h for 1st PROM check ;\
[C67Fh]*100h+7Fh   RST 28h for 1st PROM check ; for decrypting the
[C77Fh]+C700h      RST 20h for 1st PROM check ; 32h-byte area
[C77Fh]*100h+7Fh   RST 30h for 1st PROM check ;/
[D627h]+D600h      RST 38h for 2nd PROM check ;\for decrypting the
[D627h]*100h+27h   RST 28h for 2nd PROM check ; 21-byte title (and
[D727h]+D700h      RST 20h for 2nd PROM check ; verifying [DC3Fh])
[D727h]*100h+27h   RST 30h for 2nd PROM check ;/
[(where are?)]     RST's   for 3rd PROM check ;-this part looks bugged
[DC15h+00h..29h]   Spaces,FFh,"-credit play" (with underline attr) (for Menu)
[DC3Fh]            8bit chksum for 2nd PROM security check
[DD3Fh]            8bit chksum for 3rd PROM security check
[DEF1h..DEFFh]     Title (for Bookkeeping) (in 8bit OSD characters)
[DF00h..DF02h]     Token Entrypoint 1 (Goto token)
[DF05h..DF07h]     Token Entrypoint 2 (Goto token) (overlaps below DF06h!)
[[DF06h]+6]        Title Xloc+Odd MSBs (for title-centering via token 66h)
[NNNNh]            Further locations accessed via pointers in 32h-byte area
[C032h]            16bit Ptr to inst.chksum.lsb ;\all WORDs at C000..DFFF
[DFFEh]            16bit Ptr to inst.chksum.msb ;/added together
```

32h-Byte Area at [C034h]+00h..31h (encrypted via PROM data)

```
00h      Flags
           Bit0    (copied to port01h.w.bit4?, see 41B7h/676Eh) (should be 1)
           Bit1-3 Unused (should be 0) (or... Bit3 is used at 0B75h ?)
           Bit4    (checked at 077Bh) (use [1Eh] as Game Over length?)
           Bit5   Used entry (must be 1) (otherwise treated as empty slot)
           Bit6   Checksum Type ([2Ah,2Bh] and num "0" bits in chk[2Eh-2Fh])
           Bit7   Skill Mode (0=Time-Limit Mode, 1=Skill Mode)
01h      GameID (must be a unique value; BIOS rejects carts with same IDs)
02h-16h  Title (21 OSD chars) (needs second PROM decryption pass)
17h-18h  Attraction/Demo Time (in "NMI" units) ("You Are Now Viewing...")
19h-1Ah  VRAM Addr for Inserted Credits string (during game play)
1Bh-1Ch  Ptr to List of Encrypted Instruction Text Lines
         (len byte, followed by len+1 pointers to 24-word text strings)
1Dh      Default Price (number of credits per game) (LSB must be 01h..09h)
1Eh      Time Minutes (BCD) ;\(TIME mode: MUST be 01:00 .. 30:00 and LSB
1Fh      Time Seconds (BCD) ;/MUST be 0 or 5) (SKILL mode: don't care?)
20h-21h  VRAM Addr for Remaining Time value (unused in Skill Mode)
22h      SNES Watchdog (SNES must read joypads every N frames; 00h=Disable)
23h         ??? Byte... (jump enable for token 60h) (allow money-back?)
24h         ???Byte, alternate for [25h]?
25h         ???Byte, time-limit related; combined with [1Eh..1Fh,26h..27h]?
26h-27h     ???Word (unused for GameID 00-02; these use 00C0h/0140h)
28h-29h  Unused (0000h)
2Ah-2Bh  Checksum adjust (optional XOR value for [30h-31h], when Flags.6=1)
2Ch-2Dh  Encrypted.ptr to 4th check xfer.order.XOR.byte (eg.byte 07h=reverse)
2Eh-2Fh  Encrypted.ptr to 4th check 8-byte key (sometimes depends [01h])
30h-31h  Checksum accross [00h..2Fh], eventually XORed with [2Ah]:[2Bh]
```

Note: After decryption, the above 32h-bytes are stored at 8s00h..8s31h (with s=0..2 for slot 1-3).

Note: Instructions can be viewed by pressing Instructions Button, either during game, or in demo mode.

Skill Mode Notes

There are some variants (unknown how exactly to select which variant):

```
Game RESTARTS after Game Over (if one still has credits)
Game CONTINUES after Game Over (if one still has credits)
```

And, if one does NOT have credits remaining:
```
Game PROMPTS insert coin to CONTINUE (eg. ActRaiser)
Game ABORTS and goes to Game Menu (eg. Addams Family)
```
And, for supporting Skill Mode, the DF00h function must contain a Poke(8060h,00h) token.

GameID Notes
Known values used by original games are 00h..09h, FDh, and FFh. The homebrew Magic Floor game is using ID 3Fh. The no$sns/a22i tool assigns IDs 40h..BFh based on the game Title checksum (that assignment does more or less reduce risk that different homebrew games could conflict with each other).

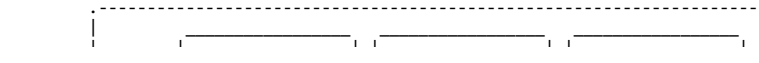
Tools
The a22i assembler (in no$sns debugger, v1.3 and up) allows to create INST ROM files with title, instructions, checksums, time/skill settings, and special PROM-less RST handlers. For details see the "magicnss.a22" sample source code in the "magicsns.zip" package.

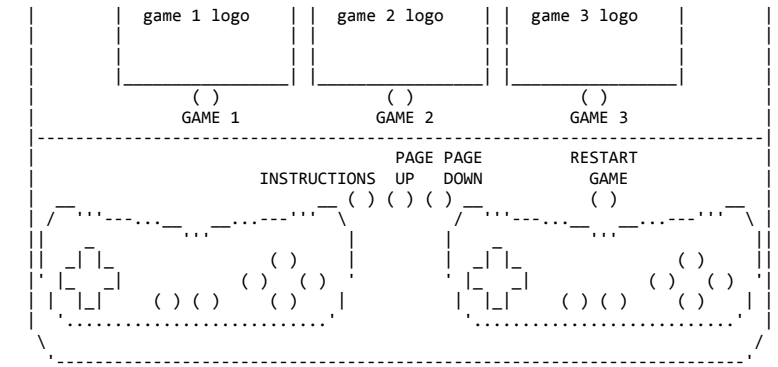
NSS Interpreter Tokens

Tokens
```
00h  Reboot_Bios()
02h  Osd_Wrstr_Direct(Len8,VramAddr16,Data16[Len], ... ,FFh,Sleep0)
04h  Osd_Wrstr_Encrypted_Txt_Line(Yloc*12,Sleep0)
06h  Osd_Wrstr_Prom_Title_Slot_80C0h(Len8-1,VramAddr+2000h*N,Sleep0)    ?
08h  Osd_Wrstr_Prom_Title(Slot+80h,Len8-1,VramAddr+2000h*N,Sleep0)      ?
0Ah  Port_00h_W_Set_Bits(OrValue)
0Ch  Port_01h_W_Set_Bits(OrValue)
0Eh  Port_03h_W_Set_Bits(OrValue)
10h  Port_00h_W_Mask_Bits(AndValue)
12h  Port_01h_W_Mask_Bits(AndValue)
14h  Port_03h_W_Mask_Bits(AndValue)
16h  Set_80C2h_To_Immediate(Imm8)
18h  Set_80C3h_To_Immediate(Imm8)
1Ah  Set_80C4h_To_Immediate(Imm8)
1Ch  Set_80C5h_To_Immediate(Imm8)
1Eh  Compare_And_Goto_If_Equal(Addr16,Imm8,Target)          ;\
20h  Compare_And_Goto_If_Not_Equal(Addr16,Imm8,Target)      ; unsigned
22h  Compare_And_Goto_If_Below_or_Equal(Addr16,Imm8,Target) ; cmp [addr],imm
24h  Compare_And_Goto_If_Above(Addr16,Imm8,Target)          ;/
26h  Decrement_And_Goto_If_Nonzero(Addr16,Target)
28h  Poke_Immediate(Addr16,Imm8)
2Ah  Sleep_Long(Sleep16)
2Ch  Disable_Interpreter_and_Reset_Gosub_Stack()
2Eh  Osd_Display_Num_Credit_Play(Slot*4,VramAddr16,Sleep0)
30h  Test_And_Goto_If_Nonzero(Addr16,Imm8,Target)
32h  Test_And_Goto_If_Zero(Addr16,Imm8,Target)
34h  Osd_Wrstr_Indirect(Addr16,Sleep0)
36h  Gosub_To_Subroutine(Target)   ;\max 3 nesting levels
38h  Return_From_Subroutine()      ;/
3Ah  Goto(Target)
3Ch    _xxx()        ... init some values
3Eh    _xxx()     ... init more, based on inst rom
40h  Wait_Vblank()          ;or so (waits for Port[00h].bit6)
42h  Osd_Wrstr_Indexed(index8,Sleep0)
44h  Reload_Attraction_Timer()
46h    _xxx()    ... advance to next instruction page ... or so
48h  Handle_PageUpDown_For_Multipage_Instructions()
4Ah  Reload_SNES_Watchdog()
4Ch  Decrease_SNES_Watchdog_and_Goto_if_Expired(Target)
4Eh    _xxx_osd_SPECIAL...(Slot+80h,Len8-1,VramAddr+2000h*N,Sleep0) ? bugged?
50h    _copy_cart_flag_bit0_to_port_01_w_bit4()
52h  Map_Slot_80C0h()
54h  Osd_Wrstr_Indirect_Encrypted(Addr16,Sleep0)
```
Below exist in BIOS version "03" only:
```
56h  Osd_Wrstr_Num_Credit_Play(VramAddr16,Sleep0)
58h  Map_Slot_804Ch()
5Ah    _xxx()      ;two lines: SubtractVramAddrBy1Ah_and_Strip_Underline ?
5Ch  Osd_Wrstr_Prom_Title_Slot_804Ch_unless_Slot1_Empty(Len8,VramAddr,Sleep0)
5Eh     Copy_8s19h_To_81E9h()      ;=VRAM Addr for Credits String
60h  Goto_If_8s23h_Nonzero(Target)
62h    _xxx(Target)       ;load timer from 8s24h or 8s25h goto if zero
64h  Goto_If_GameID_is_00h_or_01h_or_02h(Target)
66h  Create_Centered_Osd_Wrstr_Title_Function_at_84C0h(yloc*24)
68h    _xxx()           ;... 8s25h, 8s26h, and MM:SS time-limit related ?
```
And, some general token values:
```
56h..7Eh  Unused_Lockup()   ;unused version "02" tokens ;\jump to an
6Ah..7Eh  Unused_Lockup()   ;unused version "03" tokens ;/endless loop
01h..7Fh  Crash()           ;odd token numbers jump to garbage addresses
80h..FFh  Sleep_Short(Sleep7)  ;00h..7Fh (in LSBs of Token)
```
Sleep0 is an optional 00h-byte that can be appended after the Wrstr(Params) commands. If the 00h-byte is NOT there, then a Sleep occurs for 1 frame. If the 00h-byte is there, then token execution continues (after skipping the 00h) without Sleeping.

Note
INST ROM contains two interpreter functions (invoked via Gosub DF00h and Gosub DF05h).
```
DF00h - Custom code (quite simple in F-Zero, very bizarre in ActRaiser)
DF05h - Display centered & underlined Title in first line
```
Available stack depth is unknown (at least one stack level is used, so there are max two free levels, or maybe less) (the DF00h function CAN use at least one stack level).
The DF05h function is used for displaying the instructions headline (when viewing instructions in Demo mode). The purpose/usage of the DF00h function is unknown; essentially, everything works fine even if it just contains a Return token; for Skill Mode games it also seems to require a Poke(8060h,00h) token.

NSS Controls

Front Panel




Unlike normal cabinets, the NSS doesn't have arcade joysticks. Instead, there are two huge SNES joypads firmly mounted on the front-plate (about twice as big as normal SNES joypads). L/R symbols are depicted on the joypad "surface" (although the actual L/R buttons seem to be on "rear" shoulders as usually).

GAME 1..3 and INSTRUCTION buttons are fitted with LEDs
For single-cartridge use, there may be a different front-panel without GAME 1-3 buttons (there is no Game Menu, and the Config screen is joypad controlled in single-cart mode).

Plus, TEST Button, SERVICE Button
Plus, TWO Coin input/switches
Plus, DIP-Switches in Cartridge

NSS Games, BIOSes and ROM-Images

Nintendo Super System BIOS (Nintendo)
The BIOS is stored a 32Kx8 EPROM on the mainboard. There are at least three BIOS versions (the version number, "02" for oldest version, and "03" for the two newer versions, is shown at the top of the Selftest result screen). The "02" version is incompatible with newer games (works only with the 3 oldest titles).

```
NSS-v02.bin  aka NSS-C.DAT    ;CRC32: A8E202B3 (version "02" oldest)
NSS-v03a.bin aka NSS-IC14.02  ;CRC32: E06CB58F (version "03" older)
NSS-v03b.bin aka NSS-V3.ROM   ;CRC32: AC385B53 (version "03" newer/patch)
```

NSS Cartridge ROM-Images
ROM-Images should consist of following components in following order:

```
1. PRG-ROM (the SNES game) (usually 512Kbytes or 1024Kbytes)
2. INST-ROM (the Z80 title & instructions) (32Kbytes)
3. PROM (decryption key) (16 bytes)
```

Note: For the Type B/C PCBs, the PROM is 16 bytes in size. The Type A PCBs seem to be somehow different - details are still unknown; the ROM-image format may need to be changed in case that those details are discovered.
The existing cartridges don't contain any coprocessors - if somebody should make such cartridges, please insert the coprocessor ROM (eg. DSP1) between PRG-ROM and INST-ROM.

NSS Games

```
PCB Title
C   Act Raiser (NSS) 1992 Enix (Two EPROMs+DIPSW)
C   Addams Family, The (NSS) 1992 Ocean (Two EPROMs+DIPSW)
C   Contra 3: The Alien Wars (NSS) 1992 Konami (Two EPROMs+SRAM+DIPSW)
C   David Crane's Amazing Tennis (NSS) 1992 Abs.Ent.Inc. (Two EPROMs+DIPSW)
B   F-Zero (NSS) 1991 Nintendo (ROM+SRAM)
C   Irem Skins Game, The (NSS) 1992 Irem (Two EPROMs+DIPSW)
C   Lethal Weapon (NSS) 1992 Ocean (Two EPROMs+DIPSW)
-   Magic Floor (NSS) 2012 nocash (EPROM+DIPSW, works without PROM)
C   NCAA Basketball (NSS) 1992 Sculptured Software Inc. (Two EPROMs+DIPSW)
C   Robocop 3 (NSS) 1992 Ocean (Two EPROMs+DIPSW)
A   Super Mario World (NSS) 1991 Nintendo (ROM)
A   Super Soccer (NSS) 1992 Human Inc. (EPROM)
A   Super Tennis (NSS) 1991 Nintendo (ROM)
```

Additionally, Ocean has announced Push-Over (unknown if that was ever released). And, there seems to have been a Super Copa cartridge in Mexico. And, there is somebody owning a NHL Stanley Cup prototype cartridge.
Contra 3 also appears to exist as prototype only (its INST-ROM title/instructions are just saying "New Game 1" and "To be announced").

NSS Component Lists

Cartridge PCB "NSS-01-ROM-A" (1991 Nintendo)

```
IC1   32pin  PRG ROM (LH534J ROM or TC574000 EPROM) (512Kx8 LoROM)
IC2   16pin  74HC367 (2bit + 4bit drivers) (unknown purpose... for PROM?)
IC3   28pin  INST-ROM (27C256) (32Kx8 EPROM)
IC4   8pin   Key-Chip (RP5H01 serial 72bit PROM)
CL/SL 2pin   Jumpers (see notes)
CN?   100pin Cartridge connector (2x50pin)
```

Used by Super Mario World (ROM), Super Tennis (ROM), and Super Soccer (EPROM).
For ROM: Short CL1-CL5, Open SL1-SL5. For EPROM: Short SL1-SL5, Open CL1-CL5.

Cartridge PCB "NSS-01-ROM-B" (1991 Nintendo)

```
IC1   28pin  SRAM (LH5168FB-10L)
IC2   32pin  PRG ROM (LH534J ROM) (512Kx8 LoROM)
IC3   16pin  74LS139 (demultiplexer) (for ROM vs SRAM mapping)
IC4   16pin  74HC367 (2bit + 4bit drivers) (unknown purpose... for PROM?)
IC5   14pin  74HC27 (3x3 NOR) (for SW1) (not installed on the F-Zero board)
IC6   14pin  74HC10 (3x3 NAND)(for SW1) (not installed on the F-Zero board)
IC7   20pin  74HC540 (inv.drv)(for SW1) (not installed on the F-Zero board)
IC8   28pin  INST-ROM (27C256) (32Kx8 EPROM)
IC9   8pin   Key-Chip (RP5H01 serial 72bit PROM)
SW1   16pin  DIP-Switch (8 switches)    (not installed on the F-Zero board)
AR1   9pin   Resistor network (for SW1) (not installed on the F-Zero board)
```

```
  CL/SL 2pin   Jumpers (see notes)
  CN?   100pin Cartridge connector (2x50pin)
```
Used only by F-Zero. For that game: Short CL1-CL7, Open SL1-SL7. Other settings might allow to use EPROM instead ROM, or to change ROM/SRAM capacity.

Cartridge PCB "NSS-01-ROM-C" (1992 Nintendo)
Judging from low-res photos, the PCB is basically same as NSS-01-ROM-B, but with two PRG ROM chips (for double capactity). Exact components are unknown, except for a few ones:
```
  IC1   28pin  SRAM (6116, 2Kx8) (DIP24 in 28pin socket?) (Contra III only)
  IC2   32pin  PRG-ROM-1 (TC574000 EPROM) (512Kx8 LoROM, upper half)
  IC3   32pin  PRG-ROM-0 (TC574000 EPROM) (512Kx8 LoROM, lower half)
  IC4   16pin  74LS139 (demultiplexer) (for ROM vs SRAM mapping)
  IC5   16pin  74HC367 (2bit + 4bit drivers) (unknown purpose... for PROM?)
  IC6   14pin  74HC27 (3x3 NOR) (for SW1)
  IC7   14pin  74HC10 (3x3 NAND)(for SW1)
  IC8   28pin  INST ROM (27C256) (32Kx8 EPROM)
  IC9   20pin  74HC540 (inv.drv)(for SW1)
  IC10  8pin   Key-Chip (RP5H01 serial 72bit PROM)
  SW1?  16pin  DIP-Switch (8 switches)  (installed)
  AR1   9pin   Resistor network for SW1 (installed)
  BAT1? 2pin   Battery (CR2032 3V coin) (with socket) (Contra III only)
  CL/SL 2pin   Jumpers (see notes)
  CN?   100pin Cartridge connector (2x50pin)
```
Used by ActRaiser, Addams Family, Amazing Tennis, Irem Skins Game, Lethal Weapon, NCAA Basketball, Robocop 3 (all without SRAM), and, by Contra III (with SRAM). Default (for all those games) is reportedly: Short CL2-CL6,CL12-CL13,CL15,CL17-CL19, Open SL1,SL7-SL12,SL14,SL16,SL20-SL22.
DIP Switches are usually/always installed. Battery/SRAM is usually NOT installed, except on the Contra III cartridge (which has "NSS-01-ROM-C" PCB rebadged as "NSS-X1-ROM-C" with a sticker).

Mainboard NSS-01-CPU MADE IN JAPAN (C) 1991 Nintendo
Below lists only the main chipset (not the logic chips; which are mostly located on the bottom side of the PCB).
Standard SNES Chipset
```
  S-CPU 5A22-02 (QFP100)
  S-PPU1 5C77-01 (QFP100)
  S-PPU2 5C78-01 (QFP100)
  S-WRAM LH68120 (SOP64) 128Kx8 DRAM with sequential access feature (SNES WRAM)
  Fujitsu MB84256-10L 32Kx8 SRAM (SOP28) (SNES VRAM LSBs)
  Fujitsu MB84256-10L 32Kx8 SRAM (SOP28) (SNES VRAM MSBs)
```
NSS/Z80 Specific Components
```
  Zilog Z84C0006FEC Z80 CPU, clock input 4.000MHz (QFP44)
  27C256 32Kx8 EPROM "NSS-C_IC14_02" (DIP28) (Z80 BIOS)
  Sharp LH5168N-10L 8Kx8 SRAM (SOP28) (Z80 WRAM)
  Mitsubishi M50458-001SP On-Screen Display (OSD) Chip (NDIP32)
  Mitsubishi M6M80011 64x16 Serial EEPROM (DIP8)
   (Pinout: 1=CS, 2=CLK, 3=DATA IN, 4=DATA OUT, 5=VSS, 6=RESET, 7=RDY, 8=VCC)
  Seiko Epson S-3520 Real Time Clock (SOIC14)
```
Amplifiers/Converters/Battery and so
```
  Sharp IR3P32A (chroma/luma to RGB converter... what is that for???) (NDIP30)
  Hitachi HA13001 Dual 5.5W Power Amplifier IC
  Matsushita AN5836 DC Volume and Tone Control IC (SIL12)
  Mitsumi Monolithic MM1026BF Battery Controller (SOIC8) (on PCB bottom side)
  5.5V - 5.5 volt supercap
```
Oscillators
```
  21.47724MHz SNES NTSC Master Clock
  14.31818MHz (unknown purpose, maybe for OSD chip or RGB converter or so)
  4.000MHz for Z80 CPU
  32.678kHz for RTC
  <unknown clock source> for OSD Dotclock
```
Connectors
```
  "JAMMA" - 2x28 pin
  CN11/12/13 - 2x50 pin connectors for game carts
  CN2 - 10 pin connector
  CN3 - 13 pin connector
  CN4 - 8 pin connector
  CN5 - 7 pin connector
  CN6 - 24 pin connector (to APU daughterboard)
```
Jumpers
```
  SL1/SL2/SL3/CL1/CL2 - Mono/stero mode (for details see PCB text layer)
  SL4 - Use Audio+ (pin 11 on edge connector)
  SL5 - Unknown purpose
```
APU Daughterboard (shielded unit, plugged into CN6 on mainboard)
```
  Nintendo S-SMP (M) SONY (C) Nintendo '89' (QFP80) (SNES SPC700 CPU)
  Nintendo S-DSP (M) (C) SONY '89' (QFP80) (SNES sound chip)
  Toshiba TC51832FL-12 32Kx8 SRAM (SOP28) (1st half of APU RAM)
  Toshiba TC51832FL-12 32Kx8 SRAM (SOP28) (2nd half of APU RAM)
  Japan Radio Co. JRC2904 Dual Low Power Op Amp (SOIC8)
  NEC D6376 Audio 2-Channel 16-Bit D/A Converter (SOIC16)
  CN1 - 24 pin connector (to CN6 on mainboard)
  <unknown clock source> for APU (probably SNES/APU standard 24.576MHz)
```

NSS Pin-Outs

Cartridge Slots (3 slots, 2x50pin each)
All 100 pins are completely unknown. Note: Portions are probably resembling the normal 62pin SNES slot. The slot selection, PROM, and INST ROM bus might resemble the schematic in the Playchoice 10 service manual.

Big Edge Connector "JAMMA" - 2x28 pin
```
  1  GND (from Power Supply)
  A  GND (from Power Supply)
  2  GND (NC)
  B  GND (from Power Supply)
  3  +5V (from Power Supply)
  C  +5V (to joypads; and NC there)
  4  +5V (from Power Supply)
  D  +5V (from Power Supply)
  5  NC  (NC)
```

```
E  -5V (from Power Supply)
6  +12V (to Coin Lamps and Coin Counter)
F  +12V (from Power Supply)
7       KEY
H       KEY
8  Coin Counter 1
J  Coin Counter 2
9  NC
K  NC
10 SPEAKER (Right)
L  SPEAKER (Left)
11 AUDIO (+) (NC)
M  AUDIO GND
12 VIDEO RED
N  VIDEO GREEN
13 VIDEO BLUE
P  VIDEO SYNC
14 VIDEO GND
R  SERVICE SW
15 TEST SW
S  NC
16 COIN SW 1
T  COIN SW 2
17 1P START
U  2P START
18 1P UP
V  2P UP
19 1P DOWN
W  2P DOWN
20 1P LEFT
   2P LEFT
21 1P RIGHT
   2P RIGHT
22 1P A
   2P A
23 1P B
   2P B
24 1P SELECT
   2P SELECT
25 VOLUME ?   (POT Center Pin)
   VOLUME ?   (POT Outer Pin)
26 VOLUME GND (POT Outer Pin)
   NC
27 GND
   GND
28 GND
   GND
```

13P Port (Front Panel LEDs/Buttons)

```
1  GND (for Buttons)
2  Button Restart
3  Button Page Down
4  Button Page Up
5  Button Instructions
6  Button Game #3
7  Button Game #2
8  Button Game #1
9  LED Instructions
10 LED Game #3
11 LED Game #2
12 LED Game #1
13 +5V or so (for LEDs)
```

10P (Extra Joypad Buttons)

```
1  GND
2  2P TR
3  2P TL
4  2P Y
5  2P X
6  1P TR
7  1P TL
8  1P Y
9  1P X
10 GND
```

NSS Repair (Blank Screen / Washed out colors)

There seems to be a fairly common hardware problem that causes the NSS to show a picture with washed out colors or a completely blank screen; in some cases the problem appears or disappears when the unit has warmed up.
The problem is related to the power supply of the IR3P32A chip: The supply should be around 9V, and video glitches appear when it drops below 8V. For deriving the "9V", Nintendo has strapped the IR3P32A to the 12V line via a 100 ohm resistor; which is rather crude and unreliable.
As workaround one could add a second resistor in parallel with the 100 ohms (which is equally crude, though it should help temporarily), a more reliable solution should be to replace the 100 ohms by a 7809 voltage regulator (and eventually some capacitors as far as needed).
The actual reason for the problem is unknown - apparently some odd aging effect on the IR3P32A chip and/or other components connected to it. No info if the problem occurs both with original monitor and power supply as well as with third-party hardware.

NSS On-Screen Controller (OSD)

On-Screen Display Controller M50458-001SP (Mitsubishi Microcomputers)

OSD Addresses

The OSD Address is transferred as first word (after chip select):

```
0000h..011Fh  Character RAM (24x12 tiles, aka 288 tiles, aka 120h tiles)
0120h..0127h  Configuration Registers (8 registers)
```

Further words are then written to the specified address (which is auto-incremented after each word).

Character Codes (for OSD Address 000h..011Fh)

```
  0-6   Character Number (non-ASCII)
  7     Unused (zero)
  8-10  Text Color     (on NSS: 3bit RGB) (Bit0=Red, Bit1=Green, Bit2=Blue)
  11    Blinking flag  (0=Normal, 1=Blink)
  12    Underline flag (0=Normal, 1=Underline)
  13-15 Unused (zero)  (on NSS: used as hidden PROM check flags by NSS BIOS)
```

The M50458-001SP charset has been dumped by DogP, letters & punctuation marks are:

```
  Character  <---00h..0Fh---><---10h..1Fh---><---20h..2Fh---><---30h..3Fh--->
  00h..3Fh  "0123456789-:/.,'ABCDEFGHIJKLMNOPQRSTUVWXYZ[]();?| "
  40h..7Fh  "_abcdefghijklmnopqrstuvwxyz+*=# "
```

All characters are 12x18 pixels in size.

OSD M50458 Register 0 - Port Output Control

```
  0     P0 Usage (0=Manual Control, 1=YM; Luminance)
  1     P1 Usage (0=Manual Control, 1=BLNK; Blanking)
  2     P2 Usage (0=Manual Control, 1=B; Blue)
  3     P3 Usage (0=Manual Control, 1=G; Green)
  4     P4 Usage (0=Manual Control, 1=R; Red)
  5     P5 Usage (0=Manual Control, 1=CSYN; Composite Sync)
  6-11  Manual P0-P5 Output Level (0=Low, 1=High)
  12    Synchronize Port Output with Vsync (0=No, 1=Yes)
  13-15 Unused (zero)
```

NSS uses values 003Fh (whatever/maybe SNES as backdrop), and 00BDh (maybe solid backdrop).

OSD M50458 Register 1 - Horizontal Display Start/Zoom

```
  0-5   Horizontal Display Start in 4-pixel (?) units
  6-7   Horizontal Character Size in Line 1     (0..3 = 1,2,3,4 pixels/dot)
  8-9   Horizontal Character Size in Line 2..11 (0..3 = 1,2,3,4 pixels/dot)
  10-11 Horizontal Character Size in Line 12    (0..3 = 1,2,3,4 pixels/dot)
  12    PAL: Interlace Lines (0=625 Lines, 1=627 Lines) NTSC: Unused (zero)
  13-15 Unused (zero)
```

NSS uses 0018h (normal centered display) and 011Bh (fine-adjusted position in intro screen).

OSD M50458 Register 2 - Vertical Display Start/Zoom

```
  0-5   Vertical Display Start in 4-scanline (?) units
  6-7   Vertical Character Size in Line 1     (0..3 = 1,2,3,4 pixels/dot)
  8-9   Vertical Character Size in Line 2..11 (0..3 = 1,2,3,4 pixels/dot)
  10-11 Vertical Character Size in Line 12    (0..3 = 1,2,3,4 pixels/dot)
  12    Halftone in Superimpose Display (0=Halftone Off, Halftone On)
  13-15 Unused (zero)
```

NSS uses 0009h (normal centered display) and 0107h (fine-adjusted position in intro screen).

OSD M50458 Register 3 - Character Size

```
  0-4   Vertical Scroll Dot Offset (within char) (0..17) (18..31=Reserved)
  5-6   Vertical Space between Line 1 and 2 (0..3 = 0,18,36,54 scanlines)
  7     Control RS,CB Terminals (0=Both Off, 1=Both On)
  8-11  Vertical Scroll Char Offset (0=No Scroll, 1..11=Line 2-12, 12..15=Res.)
  12    PAL: Revise 25Hz Vsync (0=No, 1=Yes/Revice)  NTSC: Unused
  13-15 Unused (zero)
```

NSS uses 0000h (normal 1x1 pix size) and 082Ah (large 2x2 pix "NINTENDO" in intro), 0y20h (in-demo: instructions with double-height headline? and y-scroll on 2nd..10th line), 0y00h (in-game: instructions without headline and fullscreen scroll).
Verical Scroll OFF: Show 12 lines
Verical Scroll ON: Show 11 lines (1st line fixed, 10 lines scrolled)
(in scroll mode only 11 lines are shown)
(allowing to update the hidden 12th line without disturbing the display)

OSD M50458 Register 4 - Display Mode

```
  0-11  Display Mode Flags for Line 1..12 (0=Via BLK0,BLK1, 1=Via Different)
  12    LINEU - Underline Display (0=Off, 1=On) "depends on above bit0-bit11"
  13-15 Unused (zero)
```

NSS uses 0000h.

OSD M50458 Register 5 - Blinking and so on

```
  0-1   Blink Duty  (0=Off, 1=25%, 2=50%, 3=75%) (WHAT color during WHAT time?)
  2     Blink Cycle (0=64 Frames, 1=32 Frames)
  3     Horizontal Border Size (0..1 = 1,2 dots)
  4-5   Blink/Inverse Mode (0=Cursor, 1=ReverseChr, 2=ReverseBlink, 3=AltBlink)
           aka EXP0,EXP1 (see details below)
  6     Horizontal Display Range when all chars are in matrix-outline (0..1=?)
  7     OSCIN frequency (0=4*fsec, 1=2*fsec) (for NTSC only)
  8     Color Burst Width (0=Standard, 1=Altered)
  9     Vsync Signal separated from Composite Sync (0=No, 1=Separated Circut)
  10-12 Test Register "Exception video RAM display mode" (should be zero)
  13-15 Unused (zero)
```

NSS uses 0240h, 0241h, 0247h.

OSD M50458 Register 6 - Raster Color

```
  0-2   Raster Color    (on NSS: 3bit RGB) (Bit0=Red, Bit1=Green, Bit2=Blue)
          (aka Backdrop color?)
  3     Composite Signal BIAS (0=Internal BIAS Off, 1=Internal BIAS On)
  4-6   Character Background Color         (Bit0=Red, Bit1=Green, Bit2=Blue)
  7     Blanking Level (0=White, 1=Black)
  8-10  Cursor and Underline Display Color (Bit0=Red, Bit1=Green, Bit2=Blue)
  11    Cursor/Underline Color for Dot 1  (0=From VRAM, 1=From above bit8-10)
  12    Cursor/Underline Color for Dot 18 (0=From VRAM, 1=From above bit8-10)
  13-15 Unused (zero)
```

NSS uses 1804h, 1880h, 1882h, 1884h.

OSD M50458 Register 7 - Control Display

```
  0     Raster (backdrop?) blanking (0=By Mode;bit2-3?, 1=Whole TV full raster)
  1     Background Color Brightness for RGB (0=Normal, 1=Variable) huh?
  2-3   Mode (0=Blanking OFF, 1=Chr Size, 2=Border Size, 3=Matrix-outline Size)
           aka special meanings in conjunction with register 4 (?)
  4     Mode (0=External Sync, 1=Internal Sync)
  5     Erase RAM (0=No, 1=Erase RAM) (=clear screen?)
  6     Display Output Enable for Composite Signal (0=Off, 1=On)
```

```
  7     Display Output Enable for RGB Signal       (0=Off, 1=On)
  8     Stop OSCIN/OSCOUT (0=Oscillate, 1=Stop) (for sync signals)
  9     Stop OSC1/OSC2    (0=Oscillate, 1=Stop) (for display)
  10    Exchange External C by Internal C in Y-C Mode (0=Normal, 1=Exchange)
  11    Video Signal (0=Composite, 1=Y-C output)
  12    Interlace Enable (0=Enable, 1=Disable) (only in Internal Sync mode)
  13-15 Unused (zero)
```

NSS uses 1289h, 12A9h and 12B9h.

NSS OSD Dotclock

The OSD chip is having an unknown dotclock (somewhat higher than the SNES dotclock: 12 pixels on OSD are having roughly the same width as 8 pixels on SNES).

Blink/Underline

```
  <Register> <VramAttr> Shape
  EXP1 EXP0  EXP BLINK
  x    x     0   0      " A "             Normal
  x    x     0   1      " A " <--> "   "  Character is blinking
  0    0     1   0      "_A_"             Underlined
  0    0     1   1      "_A_" <--> " A "  Underline is blinking
  0    1     1   0      "[A]"             Inverted Character
  0    1     1   1      "[A]" <--> " A "  Inversion is blinking
  1    0     1   0      "[A]"             Inverted Character
  1    0     1   1      "[A]" <--> " A "  Inversion is blinking
  1    1     1   0      "   " <--> " A "  Character is blinking, duty swapped
  1    1     1   1      " A " <--> "_ _"  Character and Underline alternating
```

SFC-Box Overview

Main Menu

```
  Allows to select from 5 games
```

Note: If the two cartridges do contain more than 5 games in total, then the GUI is divided into two pages with 4 games, plus prev/next page option (unknown if more than 8 games are also supported).

Per Game Menu (after selecting a Game in Main Menu)

```
  1. Game Start
  2. Game Instructions
  3. Game Preview
  4. Return to Main Menu
```

Soft-Reset Feature

```
  Press L+R+Select+Start (on Joypad) --> Reset Current Game
  Press Reset Button (on SFC-Box Front Panel) --> Restart Boot Menu
```

GAME/TV Button

Allows to switch between Game & TV mode. The purpose is totally unclear... maybe it just allows to disable forwarding the Antenna-input to the RF-Out connector... but, <why> should one want to disable that?

SFC-Box Cartridges

The SFC-Box contains two special multi-game cartridges. There have been only 4 cartridges produced. The first cartridge MUST be always PSS61 (contains 3 games, plus the required GUI and 128Kbyte SRAM). The second cartridge can be PSS62, PSS63, or PSS64 (which contain 2 games each; these carts have no own SRAM, but they can share portions of the SRAM from the PSS61 cart).

SFC-Box Special ROM/EPROMs

```
  KROM 1      EPROM 64Kbytes     (HD64180 BIOS in SFC-Box console)
  GROM1-1     EPROM 32Kbytes     (Directory) (IC1 in PSS61 cart)
  GROM2-1     EPROM 32Kbytes     (Directory) (IC1 in PSS62 cart)
  GROM3-1     EPROM 32Kbytes     (Directory) (IC1 in PSS63 cart)
  GROM4-1     EPROM 32Kbytes     (Directory) (IC1 in PSS64 cart)
  ATROM-4S-0 LoROM 512Kbytes     (GUI "Attraction" Menu) (ROM5 in PSS61 cart)
  DSP1        DSP ROM 8Kbytes    (or with padding: 10Kbytes)
  MB90082     OSD ROM 9Kbytes    (OSD-Character Set in MB90082-001 chip)
```

ATROM-4S-0 contains a regular SNES header at 7FC0h, interesting entries are:

```
  7FC0h Title "4S ATTRACTION        "
  7FD6h Coprocessors (00h) (none, but, ATROM can communicate with the HD64180)
  7FD8h RAM Size (00h) (none, but, GROM indicates 32Kbytes allocated to ATROM)
  7FDAh Maker (B6h) (HAL)
```

Note: "GROM3-1" is dumped (but its ROM-image is "conventionally" misnamed as "GROM1-3"). There is reportedly also a "different" GUI version (not confirmed/details unknown, maybe there's just a configuration setting, in SRAM or EPROMs or so, that changes the GUI appearance).

SFC-Box Game ROMs

```
  SHVC-4M-1  LoROM 2048Kbytes (Mario Collection) (ROM3 in PSS61 cart)
  SHVC-MK-0  HiROM 512Kbytes  (Mario Kart)       (ROM12 in PSS61 cart)
  SHVC-FO-1  LoROM 1024Kbytes (Starfox)          (IC20 in PSS61 cart)
  SHVC-GC-0  LoROM 1024Kbytes (WaiaraeGolf)      (ROM1 in PSS62 cart)
  SHVC-2A-1  HiROM 512Kbytes  (Mahjong)          (ROM9 in PSS62 cart)
  SHVC-8X-1  HiROM 4096Kbytes (Donkey Kong)      (ROM7 in PSS63 cart)
  SHVC-T2-1  LoROM 1024Kbytes (Tetris2/Bombliss) (ROM3 in PSS63 cart)
  SHVC-8X-1  HiROM 4096Kbytes (Donkey Kong)      (ROM7 in PSS64 cart)
  SHVC-M4-0  HiROM 1024Kbytes (Bomberman2)       (ROM9 in PSS64 cart)
```

All Game ROMs seem to be identical as in normal (japanese) cartridges, (ie. without any SFC-Box specific revisions).

SFC-Box ROM-Images

ROM-Images should contain all EPROMs/ROMs from the cartridge, ordered as so:

```
  GROM + ROM0(+ROM1(+ROM2(+etc))) (+DSP1)
```

The GROM at the begin of the file does also serve as file header:

```
  The size of the GROM (1 SHL N kbytes) is found in GROM [0001h].
  The number of ROMs is found in GROM [0000h].
  Title & Size of ROM<n> can be found at [[0008h]+n*2]*1000h.
  Physical IC Socket ID for ROM<n> can be found in GROM at [0008h]+[0000h]*2+n.
  The presence of a DSP ROM Image is indicated in GROM [0004h].Bit1.
```

With that information, one can calculate the file-offsets for each ROM.

```
GROM1+ROM0+ROM1+ROM2+ROM3+DSP + GROM2+ROM0+ROM1
```
Before merging GROM+ROMs, make sure that the ROMs are raw-images (without 512-byte copier headers), and that the DSP ROM is unpadded (8Kbytes), in little-endian format.
The additional "non-cartridge" ROMs of the SFC-Box (KROM1 and MB90082) should be located in a separate BIOS folder; not in the cartridge ROM-Image.

SFC-Box Crashes
Some bugged ATROM functions (7E2125h and 7E2173h) are messing up the SNES stack, causing the SNES to run into endless execution of BRK opcodes (thereby destroying lower 8K of WRAM, any enabled SRAM, and all I/O ports). Normally, SNES emulators could stop emulation in such "beyond-repair" situations - however, for the SFC-Box, emulation must be kept running (or better: crashing), since the KROM can restore normal operation by issuing a /RESET to the SNES (for example, this is happens near completion of the "**********" progress bar in the SFC-Box boot screen). Moreover, the KROM does change SNES mapping (via Port C0h/C1h), apparently without pausing/resetting the SNES CPU during that time, thus causing SNES to execute garbage code (though there are also working situations: eg. when checking the GAME headers, the SNES executes ATROM code relocated to WRAM). And, there seems to be a situation (maybe caused by above stuff) where the SNES NMI handler jumps to Open Bus regions. Note: In most or all cases, the crashing program is running into BRK opcodes (emulating these as leaving PC and SP unchanged helps avoiding the more hazardous crash-effects).

SFC-Box Coprocessor (HD64180) (extended Z80)

This is the "heart" of the SFC-Box. The two central parts are a HD64180 CPU (with extended Z80 instruction set), and a 64Kbyte EPROM labelled "KROM 1" (HD64180 BIOS). Plus, a frightening amount of about 50 small logic chips on the mainboard & daughterboard.

Overall Features are (probably)...
```
- Injecting Controller Data (for Demo/Preview mode)
- Sniffing Controller Data (for L+R+Select+Start Soft-Reset feature)
- Send/Receive Data to the SNES Menu Program (via WRIO/RDIO ports)
- Mapping the selected Game ROM (or Menu Program) into SNES memory
- Maybe also mapping GAME-SRAM bank(s) and/or the DSP-1 chip
- Resetting the SNES, for starting the Game ROM (or Menu Program)
- Reading "GROM" data from EPROMs in the cartridges
- Reportedly drawing an extra "OSD" video layer on top of the SNES picture
- Accessing the RTC Real-Time-Clock (unknown purpose)
- Somehow logging/counting or restricting the "pay-per-play" time
- Maybe handling the GAME/TV button in whatever fashion
- Maybe handling the RESET button by software
- Maybe controlling the two GAME/TV LEDs
```

Pay-per-play
Not much known there. Some people say the SFC-Box was coin-operated... but, it doesn't contain any coin-slot, and there seem to be no external connectors for external coin-slot hardware. And, there seem to be no external connectors for a "network-cable" for automatically charging the room-bill.

Usage of [4201]=RDIO / [4213]=WRIO on SNES Side (used by Menu Program)
Default WRIO output value is 00E6h.
```
bit0 Out (usually Output=LOW, from SNES) (maybe indicate ready)
bit1 In  (data in, to SNES)
bit2 In  (status/ready/malfunction or so, to SNES)
bit3 Out (clock/ack out, from SNES)
bit4 Out (data out, from SNES)
bit5 In  (clock in, to SNES)
bit6 -   (probably normal joy1 io-line)
bit7 -   (probably normal joy2 io-line & lightgun latch)
```
After booting, the SNES menu program checks the initial "1bit" status, and does then repeatedly receive 32bit packets (one command byte, and three parameter bytes) (and, in response to certain commands, it does additionally send or receive further bytes; in some cases these extra transfers are done via joy1/joy2 shift-registers instead of via WRIO, that probably because the HLL-coded KROM is so incredibly inefficient that it needs "hardware-accelerated" serial shifts).

SFC-Box Memory & I/O Maps

most of KROM is high-level-language based crap,
this is ACTUALLY WORSE than 6502-code compiled
to run on a Z80 CPU.

Physical 19bit Memory Map
```
00000h..00FFFFh   KROM
20000h..207FFFh   WRAM (mainly 204000h..207FFFh used)
                        (area at 200000h..203FFFh used as battery-ram?
                        with read/write-protect via [A0].7?)
40000h..407FFFh   GROM-Slot 0
60000h..607FFFh   GROM-Slot 1
```
Virtual 16bit Memory Map
```
0000h..7FFFh      KROM (first 32K)
8000h..BFFFh      Bank Area (16K banks, KROM,GROM,WRAM)
C000h..FFFFh      WRAM (last 16K)
```
RAM (as used by KROM1)
```
[8000...]               <-- extra 16K RAM bank (unchanged on
                            reset/entrypoint... probably battery backed?)
...                         (that 16K are read/write-protected via [A0].7 ?)
[C000..FFFF]  work ram
```

I/O Map
```
[00h..3Fh]  HD64180 (CPU on-chip I/O ports)
[40h..7Fh]  Unused (reading returns FFh)
[80h].R     Keyswitch and Button Inputs
[80h].W     SNES Transfer and Misc Output
[81h].R     SNES Transfer and Misc Input
[81h].W     Misc Output
[82h].R/W   Unknown/unused
[83h].R     Joypad Input/Status
[83h].W     Joypad Output/Control
[84h].R/W   Joypad 1, MSB (1st 8 bits) (eg. Bit7=ButtonB, 0=Low=Pressed)
[85h].R/W   Joypad 1, LSB (2nd 8 bits) (eg. Bit0=LSB of ID, 0=Low=One)
[86h].R/W   Joypad 2, MSB (1st 8 bits) (eg. Bit7=ButtonB, 0=Low=Pressed)
[87h].R/W   Joypad 2, LSB (2nd 8 bits) (eg. Bit0=LSB of ID, 0=Low=One)
```

```
  [88h..9Fh]  Unused (mirrors of Port 80h..87h)
  [A0h].R     Real Time Clock Input
  [A0h].W     Real Time Clock Output
  [A1h..BFh]  Unused (mirror of Port A0h)
  [C0h].R     Unknown/unused (reading returns FFh)
  [C0h].W     SNES Mapping Register 0
  [C1h].R     Unknown/unused (reading returns FFh)
  [C1h].W     SNES Mapping Register 1
  [C2h..FFh]  Unused (maybe mirrors of Port C0h..C1h) (reading returns FFh)
```

16bit I/O Space (when address MSB=xx=nonzero):

```
  [xx00h..xx7Fh]  Unused (reading returns FFh) (no mirror of 0000h..003Fh)
  [xx80h..xxBFh]  Mirror of 0080h..00BFh
  [xxC0h..xxFFh]  Unknown (probably mirror of 00C0h..00FFh)
```

SFC-Box I/O Ports (Custom Ports)

[80h].R - Keyswitch and Button Inputs

```
  0   Switch Pin0 Position ("OFF") Play Mode? (2nd from left) (0=Yes, 1=No)
  1   Switch Pin1 Position ("ON")  Play Mode? (3rd from left) (0=Yes, 1=No)
  2   Switch Pin2 Position ("2")   Play Mode? (4th from left) (0=Yes, 1=No)
  3   Switch Pin3 Position ("3")   Self-Test  (5th from left) (0=Yes, 1=No)
  4   Switch Pin9 Position ("1")   Options    (1st from left) (0=Yes, 1=No)
  5   Switch Pin4 Position (N/A)   Relay Off? (6th from left) (0=Yes, 1=No)
  6   TV/GAME Button (0=On, 1=Off)
  7   RESET Button   (0=On, 1=Off)
```

[80h].W - SNES Transfer and Misc Output

```
  0   SNES Transfer STAT to SNES  (Bit2 of WRIO/RDIO on SNES side)
  1   SNES Transfer CLOCK to SNES (Bit5 of WRIO/RDIO on SNES side)
  2   SNES Transfer DATA to SNES  (Bit1 of WRIO/RDIO on SNES side)
  3   Unknown/unused
  4     ?? pulsed while [C094] is nonzero (0370h timer0 steps)
  5     ??         PLENTY used (often same as bit7)
  6   Unknown/unused
  7     ??                     (often same as bit5)
```

[81h].R - SNES Transfer and Misc Input

```
  0   Int0 Request (Coin-Input, Low for 44ms..80ms) (0=IRQ, 1=No)
  1   SNES Transfer ACK from SNES  (Bit3 of WRIO/RDIO on SNES side)
  2   SNES Transfer DATA from SNES (Bit4 of WRIO/RDIO on SNES side)
  3   Boot mode or so (maybe a jumper, or watchdog-flag, or Bit0 of WRIO/RDIO?)
  4   Unknown/unused (0) ;\joy1/slot0 or so, used by an UNUSED function (08A0h)
  5   Unknown/unused (0)  ;/(for "joy2/slot1" or so, use [A0].4-5)
  6   Int1 Request (Joypad is/was accessed by SNES or so?) (0=IRQ, 1=No)
  7   Vblank, Vsync, or Whatever flag (seems to toggle at 100..200Hz or so?)
```

[81h].W - Misc Output

```
  0   SNES Reset CPU/PPU/APU/GSU/DSP1 or so (0=Reset, 1=Normal)
  1     ??         PLENTY used
  2     ??  something basic, ATROM related (or maybe HALT snes CPU?)
  3   Int1 Acknowledge (Joypad related) (0=Ack, 1=Normal)
  4     ??         PLENTY used
  5     ?? set to 1-then-0 upon init (maybe ACK/RESET something?)
  6   Watchdog Reload (must be pulsed during mem tests/waits/transfers/etc)
  7   OSD Chip Select (for CSI/O) (0=No, 1=Select)
```

[83h].R - Joypad Input/Status

```
  0   Joy2 Port [86h..87h] ready for reading (0=No, 1=Yes)     ;\Automatic
  1   Joy1 Port [84h..85h] ready for reading (0=No, 1=Yes)     ;/Reading
  2   Unknown/unused (usually/always 0)
  3   Unknown/unused (usually/always 1) (maybe joy4 Data?)
  4   Unknown/unused (usually/always 1) (maybe joy3 Data?)
  5   Joy2 Data   (0=Low, 1=High) ;\that is inverse as on SNES ;\Manual
  6   Joy1 Data   (0=Low, 1=High) ;/(where it'd be 1=Low)      ;/Reading
  7   Unknown/unused (usually/always 0)
```

[83h].W - Joypad Output/Control

```
  0   Joypad Strobe  (0=No, 1=Yes)                           ;\Manual
  1   Joypad2? Clock (0=Yes, 1=No)                           ; Reading
  2   Joypad1? Clock (0=Yes, 1=No)                           ;/
  3   Joypad Reading (0=Automatic, 1=Manual)
  4   Joypad Swap    (0=Normal, 1=Swap Joy1/Joy2)
 5-7 Unknown/unused (should be 0)
```

Not quite clear if the "Swap" feature affects... software/hardware? upon reading/writing? upon manual/automatic access?

[84h].R/W - Joypad 1, MSB (1st 8 bits) (eg. Bit7=ButtonB, 0=Low=Pressed)
[85h].R/W - Joypad 1, LSB (2nd 8 bits) (eg. Bit0=LSB of ID, 0=Low=One)
[86h].R/W - Joypad 2, MSB (1st 8 bits) (eg. Bit7=ButtonB, 0=Low=Pressed)
[87h].R/W - Joypad 2, LSB (2nd 8 bits) (eg. Bit0=LSB of ID, 0=Low=One)

2x16bit Joypad data from Controller / to SNES. In Automatic Reading mode, data is automatically forwarded to SNES, and if desired, it can be read from [84h..87h] (when [83h].0-1 are zero). The clock source for reading is unknown (maybe it comes from the SNES, so it'd work only IF the SNES is reading).
In Manual Reading mode, joypad can be read via [83h], and data can be then forwarded to SNES by writing to [84h..87h].
Notes: Observe that the bits are inverse as on SNES (where it'd be 1=Low). Aside from [83h..87h], joypad seems to also somehow wired to INT1 interrupt (and [81h].R.Bit6 and [81h.W.Bit3). Also observe that [84h/86h] are containing the MSBs (not LSBs). The KROM1/ATROM are also mis-using [84h..87h] for general-purpose "high-speed" data transfers (that is, faster than the crude HLL coded software transfers in KROM1).

[A0h].R - Real Time Clock Input (S-3520)

```
  0   RTC Data In      (0=Low=Zero, 1=High=One)
  1   Unknown/unused (usually/always 0)
  2   Unknown/unused (usually/always 0)
  3   Unknown/unused (usually/always 1)
  4   Unknown/unused (0) ;\joy2/slot1 or so, used by an UNUSED function (08A0h)
  5   Unknown/unused (0) ;/(for "joy1/slot0" or so, use [81].4-5)
  6   Unknown/used?! (usually/always 1)   used/flag ?   extra BUTTON ?
```

[A0h].W - Real Time Clock Output (S-3520)
```
0   RTC Chip Select  (0=High=No,   1=Low=Select)
1   RTC Direction    (0=Low=Write, 1=High=Read)
2   RTC Data Out     (0=Low=Zero,  1=High=One)
3   RTC Serial Clock (0=Low=Clk,   1=High=Idle)
4     ??     cleared after "C632" offhold (5 timer1 steps)
5   Unknown/Set to 0 (can be changed via 0A2Dh)
6   Unknown/Unused   (can be changed via 0A26h)
7   Unlock access to lower 16K of WRAM (0=Lock, 1=Unlock) (save area)
```

[C0h].W - SNES Mapping Register 0
```
0-1 ROM Socket  (0=ROM5, 1=ROM1/7/12, 2=ROM3/9, 3=IC20)
2   ROM Slot    (0=Slot0, 1=Slot1)
3   SRAM Enable (0=Disable, 1=Enable)
4   SRAM Slot   (0=Slot0, 1=Slot1)
5   DSP Enable  (0=Disable, 1=Enable)
6   DSP Slot    (0=Slot0, 1=Slot1)
7   ROM, DSP, and/or SRAM Mapping (0=LoROM, 1=HiROM)
```

[C1h].W - SNES Mapping Register 1
```
0-1 ROM, DSP, and/or SRAM Mapping (0=Reserved, 1=GSU, 2=LoROM, 3=HiROM)
2-3 SRAM Base   (in 32Kbyte units) (range 0..3)
4   GSU Slot    (0=Slot0, 1=Slot1)
5   Zero/Unused?
6-7 SRAM Size   (0=2K, 1=8K, 2=Reserved, 3=32K)
```

[82h].R/W - Unknown/unused
[C0h].R - Unknown/unused
[C1h].R - Unknown/unused
Not used by KROM1 (nor GROMs). Reading from these ports usually/always returns FFh.

SFC-Box I/O Ports (HD64180 Ports)

System Clock
The CPU and Timer/Baudrate-Prescalers are clocked at PHI=4.608MHz (derived from a 9.216MHz oscillator, and internally divided by 2 in the HD64180).

asci_ch0 - implemented, tx is "used" for UNUSED joypad recording
asci_ch1 - implemented, but rx+tx both unused
```
initialized to 8N1 with baudrate 28.8 kbit/s (PHI/10/16 SHR 0)
```

csio - implemented, tx is used - OSD video chip
```
initialized to 230.4 kbit/s (PHI/20 SHR 0)
chipselect is controlled via port [81h].W.Bit7 (1=select, 0=deselect)
the OSD chip is having an unknown dotclock (higher than the SNES)
(12 pixels on OSD are having roughly the same width as 8 pixels on SNES)
```

timer0
```
timer0 should run at 4.608MHz/20/130 --> 1772.3 Hz
```

timer1
```
timer1 should run at 4.608MHz/20/3840 --> 60.0 Hz
```

external interrupts
```
reset  power-up, and maybe watchdog? (but, probably not "RESET" button?)
nmi    unknown/unused
int0   coin ? (must be low for 78..140 timer0 ticks) (44ms..80ms)
int1   joypad is/was accessed by snes ?
int2   unknown/unused
```

CPU Registers
```
Port Name      Expl.                                  (On Reset)
[00] CNTLA0    ASCI Channel 0 Control Reg A           (10h, bit3=var)
[01] CNTLA1    ASCI Channel 1 Control Reg A           (10h, bit3=var)
[02] CNTLB0    ASCI Channel 0 Control Reg B           (07h, bit7/bit5=var)
[03] CNTLB1    ASCI Channel 1 Control Reg B           (07h, bit7=var)
[04] STAT0     ASCI Channel 0 Status Register         (00h, bit1/2=var)
[05] STAT1     ASCI Channel 1 Status Register         (02h)
[06] TDR0      ASCI Channel 0 Transmit Data Register
[07] TDR1      ASCI Channel 1 Transmit Data Register
[08] RDR0      ASCI Channel 0 Receive Data Register
[09] RDR1      ASCI Channel 1 Receive Data Register

[0A] CNTR      CSI/O Control Register                    (0Fh)
[0B] TRDR      CSI/O Transmit/Receive Data Register

[0C] TMDR0L    Timer 0 Counter "Data" Register, Bit0-7   (FFh)
[0D] TMDR0H    Timer 0 Counter "Data" Register, Bit8-15  (FFh)
[0E] RLDR0L    Timer 0 Reload Register, Bit0-7           (FFh)
[0F] RLDR0H    Timer 0 Reload Register, Bit8-15          (FFh)
[10] TCR       Timer Control Register                    (00h)
[14] TMDR1L    Timer 1 Counter "Data" Register, Bit0-7   (FFh)
[15] TMDR1H    Timer 1 Counter "Data" Register, Bit8-15  (FFh)
[16] RLDR1L    Timer 1 Reload Register, Bit0-7           (FFh)
[17] RLDR1H    Timer 1 Reload Register, Bit8-15          (FFh)

[18] FRC       Free Running Counter (not used by SFC-Box)  (FFh)
[20-31] (DMA)  DMA Registers        (not used by SFC-Box)
[36] RCR       Refresh Control Reg  (not used by SFC-Box)  (FCh)
[3F] ICR       I/O Control Register (not used by SFC-Box)  (1Fh)
[11-13]        Reserved             (not used by SFC-Box)
[19-1F]        Reserved             (not used by SFC-Box)
[35]           Reserved             (not used by SFC-Box)
[37]           Reserved             (not used by SFC-Box)
```

```
  [3B-3E]       Reserved            (not used by SFC-Box)

  [32] DCNTL    DMA/WAIT Control Register                 (F0h)
  [33] IL       Interrupt Vector Low Register             (00h)
  [34] ITC      INT/TRAP Control Register                 (39h)
  [38] CBR      MMU Common Base Register (Common Area 1)  (00h)
  [39] BBR      MMU Bank Base Register (Bank Area)        (00h)
  [3A] CBAR     MMU Common/Bank Area Register             (F0h)
```

OSD_INIT

```
  OUT[81h]=00h                ;osd chip deselect
  OUT[0Ah]=00h                ;init CSIO
  for i=1 to 4,OUT[81H]=80h,OUT[81H]=00h,next  ;osd wake-up from reset-state
```

OSD_SEND_CMD

```
  ;in: HL=param10bit, A=(80h OR cmd*8)
  SHL  L    ;move bit7 to cy
  RCL  H    ;shift-in cy
  SHR  L    ;undo SHL (now bit7=0 for second byte)
  OR   A,H  ;merge command and 3bit data
  CALL osd_send_byte_a
  LD   A,L  ;7bit data
  JMP  osd_send_byte_a
```

OSD_SEND_BYTE

```
  set OUT[81h]=80h              ;osd chip select
  set OUT[0Bh]=data             ;prepare TX data
  set OUT[0Ah]=10h              ;start TX
  wait until (IN[0Ah] AND 10h)=0  ;wait until TX ready
  set OUT[81h]=00h              ;osd chip deselect
```

SFC-Box OSD Chip (On-Screen Display Controller)

OSD Command Summary

```
  CMD, First Byte (Command+Data)   Second Byte (More Data)  Function
  BASE  b7 b6 b5 b4 b3 b2 b1 b0    b7 b6 b5 b4 b3 b2 b1 b0
  ---  +--+-----------+--------+  +--+--------------------+ -----------------
  0 80 |1 |0  0  0  0 |FL A8 A7|  |0 |A6 A5 A4 A3 A2 A1 A0| Preset VRAM Addr
  1 88 |1 |0  0  0  1 |D2 D1 D0|  |0 |C2 C1 C0 BS B2 B1 B0| Select Color
  2 90 |1 |0  0  1  0 |AT -  M7|  |0 |M6 M5 M4 M3 M2 M1 M0| Write Character
  3 98 |1 |0  0  1  1 |S2 S1 S0|  |0 |SC SC SC -  SB SB SB| Sprite Ctrl 1
  4 A0 |1 |0  1  0  0 |IE IN EB|  |0 |MM CM MP NP -  -  DC| Screen Ctrl 1
  5 A8 |1 |0  1  0  1 |LP DM SG|  |0 |FM SV SD -  W2 W1 W0| Screen Ctrl 2
  6 B0 |1 |0  1  1  0 |BK G1 G0|  |0 |BC VD DG N3 N2 N1 N0| Line Control
  7 B8 |1 |0  1  1  1 |EC XE FO|  |0 |-  -  Y4 Y3 Y2 Y1 Y0| Vertical Offset
  8 C0 |1 |1  0  0  0 |SC XS FC|  |0 |-  X5 X4 X3 X2 X1 X0| Horizontal Offset
  9 C8 |1 |1  0  0  1 |-  -  - |  |0 |-  -  -  -  -  -  - | Reserved
  A D0 |1 |1  0  1  0 |XC XB RA|  |0 |R2 R1 R0 RS U2 U1 U0| Set under-color
  B D8 |1 |1  0  1  1 |-  -  - |  |0 |-  -  -  -  -  -  - | Reserved (Used?)
  C E0 |1 |1  1  0  0 |-  XC XC|  |0 |XC XC XC XD XD XD XD| Sprite Ctrl 2
  D E8 |1 |1  1  0  1 |-  YC YC|  |0 |YC YC YD YD YD YD YD| Sprite Ctrl 3
  E F0 |1 |1  1  1  0 |-  -  - |  |0 |-  -  -  -  -  -  - | Reserved
  F F8 |1 |1  1  1  1 |-  -  - |  |0 |-  -  -  -  -  -  - | Reserved
```

Note: Below descriptions are showing only the 10bit parameter values (without command bits in 1st byte, and without zero-bit in 2nd byte).

OSD Command 0 (80h) - Preset VRAM Address

```
  9    FL  Fill Mode (0=Normal, 1=Fill)
  8-5  An  Address A8-A5 (aka Bit3-0 of Y) (range 0..11)
  4-0  An  Address A4-A0 (aka Bit4-0 of X) (range 0..23)
```

OSD Command 1 (88h) - Select Color

```
  9-7  Dn  Unknown Color?   ;SFCBOX/MB90089 only, not MB90075 (per CHARACTER)
  6-4  Cn  Character Color (can be GRAYSCALE or COLOR)        (per CHARACTER)
  3    BS  Unknown (Shade?) ;MB90089 only, not MB90075/SFCBOX
  2-0  Bn  Background Color (always GRAYSCALE)       (per SCREEN or per LINE?)
```

OSD Command 2 (90h) - Write Character

```
  9    AT  Character Background (0=Normal/Transp, 1=Solid)  ;SFCBOX/MB90089
  8    0   Character Blink (0=Off, 1=Blink)   ;SFCBOX only, not MB90075/MB90089
  7-0  Mn  Character Tile Number (ASCII) (20h=Normal Space, FFh=Transp Space)
```

Before Command 2: Change the VRAM address and Character Color via Commands 0 and 1 (if needed).
Upon Command 2: The specified character is stored in VRAM (together with previously specified color attributes), VRAM address is automatically incremented (and wraps from X=23 to X=0 in next line). If Fill Mode is enabled, then Command 2 repeats until reaching the end of VRAM (Fill may take up to 1ms, do not send further commands during that time).
Writes aren't performed during /HSYNC period (approx 3us), as a simple workaround, configure serial access rate so that an 8-bit transfer takes more than 3us.

OSD Command 4 (A0h) - Screen Control 1

```
  9    IE  Internal/External Sync (0=Internal/Color, 1=External/Mono)
  8    IN  Interlace Mode (0=On, 1=Off)
  7    EB  Unknown (EB)     ;MB90089 only, not MB90075/SFCBOX
  6    MM  Unknown (MM)     ;MB90089 only, not MB90075/SFCBOX
  5    CM  Color/Monochrome (0=Mono, 1=Color) (affects Character + Undercolor)
  4    MP  Unknown (MP)     ;MB90089 only, not MB90075/SFCBOX
  3    NP  NTSC/PAL Mode    (0=NTSC, 1=PAL)
  2-1  0   Reserved (should be 0)
  0    DC  Display Enable   (0=Backdrop, 1=Backdrop+Background+Characters)
```

xxx pg17 - details in IE

OSD Command 5 (A8h) - Screen Control 2

```
  9    LP  Unknown    ;MB90089 only, not MB90075/SFCBOX
  8    DM  Unknown    ;MB90089 only, not MB90075/SFCBOX
  7    SG  Unknown    ;MB90089 only, not MB90075/SFCBOX
  6    FM  Unknown    ;MB90089 only, not MB90075/SFCBOX
  5    SV  Unknown    ;MB90089 only, not MB90075/SFCBOX
  4    SD  Unknown    ;MB90089 only, not MB90075/SFCBOX
```

```
  3    -   Reserved (should be 0)
  2-0  Wn  Line Spacing    ;SFCBOX/MB90089 only, not MB90075
```
Used by SFC-Box.

OSD Command 6 (B0h) - Line Control
```
  9    BK  Background Type (0=Bordered, 1=Solid 12x18)        ;-per LINE
  8    G1  Character Y-Size  (0=Normal/18pix, 1=Zoomed/36pix) ;-per LINE
  7    G0  Character X-Size  (0=Normal/12pix, 1=Zoomed/24pix) ;-per LINE
           (old MB90075: G0=Unused, G1=Affects both X+Y Size)
  6    BC  Background Control (0=Transparent, 1=Displayed)    ;-per LINE
  5    VD  Analog VOUT,YOUT,COUT Video Enable (0=Off, On)     ;\per SCREEN
  4    DG  Digital VOC2-VOC0,VOB Video Enable (0=Off, On)     ;/
  3-0  Nn  Vertical Line Number (N3-N0) (range 0..11) <-- for per LINE bits
```
Note: On the screen, a double-height line at Line Y extends through Line Y and Y+1, drawing does then continue reading the next VRAM source data from line Y+2 (ie. line Y+1 is NOT drawn).

OSD Command 7 (B8h) - Vertical Offset
```
  9    EC  Output on /HSYNC Pin (0=Composite Video, 1=Hsync)
  8    XE  Unknown (XE)     ;SFCBOX/MB90089 only, not MB90075
  7    FO  Output on FSCO Pin (0=Low, 1=Color Burst)  ;SFCBOX/MB90089 only   6    0   Reserved (should be 0)
  5    Y5  MSB of Yn?       ;SFCBOX only, not MB90075/MB90089
  4-0  Yn  Vertical Display Start Position (Y4-Y0) (in 2-pixel steps)
```
Vertical Display Start is at "Y*2+1" lines after raising /VBLK. Whereas, raising /VBLK is 15h (NTSC) or 20h (PAL) lines after raising /VSYNC).
SFC-Box seems to use a 6bit offset (so maybe MB90082 has 1-pixel steps). SFC-Box writes totally bugged values to bit7-9 (writes them to bit8-10, and replaces them by HORIZONTAL bits when changing the VERTICAL setting).

OSD Command 8 (C0h) - Horizontal Offset
```
  9    SC  Input on /EXHSYN Pin (0=Composite Video, 1=Hsync)
  8    XS  Unknown (XS)     ;SFCBOX/MB90089 only, not MB90075
  7    FC  Input on /EXHSYN Pin (0=Use3usFilter, 1=NoFilter)  ;MB90089 only
  6-5  0   Reserved (should be 0)
  5    X5  MSB of Xn        ;SFCBOX/MB90089 only, not MB90075
  4-0  Xn  Horizontal Display Start Position (X4-X0)
```
MB90075: Horizontal Display Start is at "(X+15)*12" dots after raising /HSYNC.
MB90082: Horizontal Display Start is at "(X+?)*?" dots after raising /HSYNC.
MB90089: Horizontal Display Start is at "(X+?)*3" dots after raising /HSYNC.

OSD Command A (D0h) - Set Under-color
```
  9    XC  Unknown (XC)     ;MB90089 only, not MB90075  ;sth on SFCBOX?
  8    XB  Unknown (XB)     ;MB90089 only, not MB90075  ;sth on SFCBOX?
  7    RA  Unknown (RA)     ;MB90089 only, not MB90075/SFCBOX
  6-4  Rn  Unknown (R2-R0)  ;MB90089 only, not MB90075/SFCBOX
  3    RS  Unknown (RS)     ;MB90089 only, not MB90075/SFCBOX
  2-0  Un  Under Color (U2-U0) (aka Backdrop (and Border?) color)
```
Under Color can be COLOR or GRAYSCALE (select via CM bit). Under Color is shown only in INTERNAL sync mode.

OSD Command B (D8h) - Reserved (Used?)
This, in SFCBOX and MB90092, is similar to "Sprite Control 1" in MB90089 ???
```
  9    -   Unknown (unused by SFC-Box)    ;not MB90075, not MB90089
  8    ?   Unknown (used by SFC-Box)      ;not MB90075, not MB90089
  7    -   Unknown (unused by SFC-Box)    ;not MB90075, not MB90089
  6-4  ?   Unknown (used by SFC-Box)      ;not MB90075, not MB90089
  3    -   Unknown (unused by SFC-Box)    ;not MB90075, not MB90089
  2-0  ?   Unknown (used by SFC-Box)      ;not MB90075, not MB90089
```
Used by SFC-Box. Is that a MB90082-only feature? The SFC-Box software contains a function for setting a 1bit flag, and two 3bit parameters (however, it's clipping the "3bit" values to range 0..3); the function is used only once (with flag=00h, and with the other two parameters each set to 01h, and "unused" bits set to zero).

OSD Command 3 (98h) - Sprite Control 1 (TileNo, Char/BG Colors?)
OSD Command C (E0h) - Sprite Control 2 (Horizontal Position?)
OSD Command D (E8h) - Sprite Control 3 (Vertical Position?)
```
  Unknown    ;MB90089 only, not MB90075
```
Not used by SFC-Box. According to the poor MB90089 data sheet, TileNo seems to select char "8Fh+(0..7)*10h". And X/Y coordinates seem to consist of character coordinates & pixel-offsets within that character cell.

OSD Wake-Up from Reset
After Power-on, the OSD chip is held in Reset-state (with IE=0 and DC=0). To wake-up from that state, issue four /CS=LOW pulses.

OSD Video RAM (VRAM)
Main VRAM is 288 cells (24x12), each cell contains:
```
  8bit Character Tile Number
  3bit Character Color
  probably also 1bit "AT" flag (on chips that do support it)
  plus maybe some more per-character stuff
```
Additionally, there's an array with 12 per-line settings:
```
  .. zoom bit(s)
  plus maybe some more per-line stuff
```
Other settings (like background & backdrop colors) are per-screen only.

SFC-Box Character/Outline/Background Styles
```
  AT=0      --> draw Background transparent (=Undercolor, or TV layer)
  AT=1      --> draw Background solid by using "Dn" Unknown Color
  AT=0, BK=1, BC=1    --> draw Background solid by "Bn" color (unless Char=FFh)
  AT=x, BK=0, BC=1    --> draw Outline by "Bn" color
```

OSD Character Generator ROM (CGROM)
There are 256 characters in CGROM. The Character Set is undocumented, it seems that one can order chips with different/custom character sets, chips with suffix -001 in the part number are probably containing some kind of a "standard" charset.
That "standard" charset contains normal ASCII characters (uppercase & lowercase, with normal ASCII codes eg. 41h="A", but some missing chars like "@|\"), plus japanese symbols, and some graphics symbols (volume bar, AM, PM, No, Tape and arrow symbols).
Character FFh is said to be a "blank/end" code, the meaning there is unknown. "Blank" might refer to a normal SPACE, but maybe with BG forced to be transparent (even when using solid BG). "End" might be nonsense, or maybe it forces the remaining chars (in the current line, and/or following lines) to be

OSD Color Table
Color Table (according to MB90075 datasheet):
```
  Value   0     1     2     3       4      5      6      7
  Color   Black Blue  Red   Magenta Green  Cyan   Yellow White
  Mono    Black     ... Increasing  Gray Levels ...      White
```
(ie. Colors are RGB with Bit0=Blue, Bit1=Red, Bit2=Green)
```
  Characters      --> Grayscale or Color (depending on CM bit)
  Background      --> Always Grayscale
  Backdrop        --> Grayscale or Color (depending on CM bit)
```
Colors should be enabled (via CM bit) only in INTERNAL sync mode.

OSD Datasheets / Application Manuals
```
  MB80075 commands described on 9 pages  ;\these are all 24x12 cells
  MB80082 no datasheet exists?           ; (with extra features in
  MB80089 commands summarized on 1 page  ;/MB80082 and MB80089)
  MB80092 commands described on .. pages ;-similar/newer chip or so
  MB80050 commands summarized on 1 page  ;-different/newer chip or so
```

MB90092
```
  pg80 and up
```

MB90089
```
  vram 24x12 cells (288x216 pixels)
  cgrom (character generator rom) (256 chars, 12x18 pixels)
  base.x  3pix (1/4 character)  ;\relative to END of vblank/hblank
  base.y  2 pix                 ;/
  8 color/grayscales
  shaded (3d) or bordered chars or solid bg
  8 custom chars, one can be displayed
  colors only in INTERNAL sync mode
  mono in EXTERNAL sync mode
  zoom.x / zoom.y  12x18 pix ---> 24x36 pix
  pg10 --> background
  pg11 --> shade
  pg11 --> sprite
  pg11 --> base xy   (6bit x, 5bit y) line space 0..7
  pg12 --> serial access
  pg13 --> COMMANDS
  pg15 --> wake-up
  dot clock (pins EXD,XD) can be 6MHz .. 8MHz (=affects horizontal resolution)
```
FFh is "blank" or "end" code?
display details pg10,11

MB90075
```
  base.x is only 5bit (not 6bit)
  zoom affects BOTH x+y (cannot be done separately)
  no SHADED drawing
  vram fill function ?
  CMD 01 --> without D2..D0
  CMD 02 --> without AT
  CMD 03,0C,0D --> reserved (no sprite functions)
  CMD 04 -->
  CMD 05 --> reserved (no screen ctrl 2)
  CMD 06 --> lacks G0 (but DOES have G1 ?)
  CMD 07 --> lacks XE,FO
  CMD 08 --> lacks XS,FC,X5
  CMD 0A --> only U2..U0 (lacks XC,XB,RA,R2,R1,R0,RS)
```
details on pg14..21

SFC-Box GROM Format

All SFC-Box Cartridges are containing a 32Kbyte "GROM" EPROM, the chip contains info about the ROMs in the cartridge (title, instructions, etc).

GROM Overall Memory Map
```
  0000h - Root Header and HD64180 Code
  1000h - ROM File Info Block(s)  ;usually at 1000h,2000h,3000h,etc.
  7FFCh - Checksum (above bytes at 0000h..7FFBh added together)
  7FFEh - Complement (same as above Checksum, XORed by FFFFh)
```
The various unused locations are usually FFh-filled. Checksum/Complement are located at the end of the EPROM (7FFCh in case of the usual 32Kbyte EPROMs).

GROM - Root Header (located at 0000h)
```
  0000h 1   Number of ROMs (01h..08h) (usually 02h or 04h) (NumROMs)
  0001h 1   GROM size (1 SHL N) kbytes (usually 05h=32Kbytes) (FFh=None)
  0002h 1   Unknown (00h or 01h or 09h)
  0003h 1   Unknown (00h)
  0004h 1   Chipset (07h or 00h) (Bit0:SRAM, Bit1:DSP, Bit2:GSU?)
  0005h 1   Unknown (01h or 00h)   Menu Flag?
  0006h 2   Offset to HD64180 code  (usually 0020h or 0030h)
  0008h 2   Offset to ROM Directory (usually 0010h)
  000Ah 2   Unknown (0000h)
  000Ch 1   Unknown (78h) (aka 120 decimal)
  000Dh 1   Unknown (B0h or 00h) (theoretically unused... but isn't FFh)
  000Eh 2   Unknown (FFFFh) (probably unused)
```
ROM Directory (usually located at 0010h):
```
  NumROM words  Offset to ROM info, div 1000h (usually 0001h..NumROMs)
  NumROM bytes  Physical Socket on PCB (usually 00h..03h or 01h..02h)
```
The above byte-values (usually located at 0010h+NumROMs*2) can have following known values:
```
  00h ROM5            (upper-right) (AttractionMenu)
  01h ROM1/ROM7/ROM12 (lower-right or upper-left) (MarioKar,Mahjong,Donkey)
  02h ROM3/ROM9       (middle-right) (MarioCol,Waiarae,Tetris)
  03h IC20            (special GSU ROM location) (StarFox)
```
HD64180 code (usually at 0020h or 0030h): Around 256-bytes of HD64180 code, called with following parameters:

```
BC=ptr to 10-bytes
E=same as [BC+5]
[BC+0]  ROM Slot (0 or 1)                              ;Cartridge Slot 0-1
[BC+1]  ROM Socket (0..3)                              ;from GROM[8]
[BC+2]  Mapmode (0=LoROM, 1=HiROM, 2=GSU)              ;from GROM[P0+16h]
[BC+3]  Used Chipset (bit0=SRAM, bit1=DSP)             ;from GROM[P0+2Ah]
[BC+4]  SRAM Size (0=None, 1=2K, 3=8K, 5=32K)          ;from GROM[P0+17h]
[BC+5]  SRAM Base (0..3) (0..7 when 2 chips)           ;from GROM[P0+1Ch]
[BC+6]  Slot 1 Chipset (bit0=SRAM, bit1=DSP, bit2=GSU?)  ;from Slot1.GROM[4]
[BC+7]  Slot 0 Chipset (bit0=SRAM, bit1=DSP, bit2=GSU?)  ;from Slot0.GROM[4]
[BC+8]  Copy of Port[C0h]  ;\the function must update these values alongside
[BC+9]  Copy of Port[C1h]  ;/with the new values written to Port C0h/C1h
```

Note: During execution, the 1st 16K of GROM are mapped to 8000h..BFFFh.

GROM - ROM File n Info (located at 1000h,2000h,3000h,etc.)

```
x000h 2  P0 Offset to ASCII Title and Configuration (usually 000Eh)
x002h 2  P1 Offset to Bitmap-Title-Tiles  ;\bitmap 128x24 pix (16x3 tiles)
x004h 2  P2 Offset to Bitmap-Padding-Tile ; padded to 160x24 pix (20x3 tiles)
x006h 2  P3 Offset to Bitmap-Palette      ;/with 16-color (4bpp) palette
x008h 2  P4 Offset to Shift-JIS Instruction Pages
x00Ah 2  P5 Offset to Demo-Joypad-Data (for demo/preview feature)
x00Ch 2  P6 Offset to Unused-Joypad-Data  ;<-- not included in Attraction ROM
```

ASCII Title and Configuration Field (at P0):

```
00h 22 ASCII Title (uppercase ASCII, 22 bytes, padded with spaces)
16h 1  ROM/SRAM mapping/speed? (00h=SlowLoROM, 01h=FastHiROM, 02h=GSU/NoSRAM)
17h 1  SRAM Size (1 SHL N Kbytes) (but for Menu: ATROM Header claims NoSRAM?)
18h 1  Coprocessor is DSP1 (00h=No, 01h=Yes)
19h 1  ROM Size (in 1MBit units, aka in 128Kbyte Units)
1Ah 1   Unknown (01h or 02h or 03h)
1Bh 1  Demo/Preview enable (00h=Off, 01h=On)
1Ch 1  SRAM Base (0..3) (or 0..7 when SRAMs in BOTH slots) (CHANGED by KROM1)
1Dh 1  Preferred Title (00h=SNES[FFC0h]/Destroys SNES stack, 01h=GROM[P0])
1Eh 1   Unknown ("strange values") (can be edited in menu point "2-4-1:3")
1Fh 1  Always Zero (00h) (seems to be MSB of above entry) (always zero)
20h 4  Whatever (01h,00h,00h,01h=Menu or 00h,30h,30h,05h=Game)
24h 1   Unknown (00h or 01h or 02h) (maybe... num players/joypads?)
25h 1   Unknown (00h or 05h or 1Eh) (aka decimal 0,5,30)
26h 1  Game Flag (00h=Menu, 01h=Game)
27h 1   Unknown (00h or 05h or 1Eh) (aka decimal 0,5,30)
28h 1   Unknown (00h or 80h or 90h or A0h or D0h)
29h 1   Unknown (00h or 21h or 22h or 23h)
2Ah 1  Chipset (bit0=Uses SRAM, bit1=Uses DSP)     ;<-- missing in Star Fox
2Bh 22 Unknown (all 2Eh-filled)       ;<-- located at index 2Ah in Star Fox
```

Bitmap-Title-Tiles (at P1):

```
1 byte - Unknown (Should be 80h) (probably bit7=compression flag?)
2 bytes - Number of following bytes (N) (varies 02D5h..0573h) (max=600h)
N bytes - Compressed Title Bitmap (128x24 pix, 4bpp) (16x3 Tiles)
(the uncompressed bitmap consists of 3 rows of 16 bit-planed 4bpp SNES tiles)
(see below for the compression format)
```

Bitmap-Padding-Tile (at P2):

```
32 bytes - Uncompressed Padding Tile (8x8 pix, 4bpp)
(used to pad the 128x24 pix title bitmap, centered within a 160x24 pix area)
(should be usually uni-colored tile, with same color as bitmap's background)
(or, alternately, one could probably also use a "hatched" background pattern)
```

Bitmap-Palette (at P3):

```
32 bytes - 16-color Palette for Title Bitmap (words in range 0000h..7FFFh)
(color 0 is unused/transparent, usually contains 0038h as dummy value)
```

Shift-JIS Instruction Pages (at P4):

```
1 byte - Number of Pages
N bytes - Page(s) ;max 1372 bytes per page  ;21 lines = max 6+(32*2+1)*21+1
Each page starts with a 6-byte header (usually 8,2,4,1,4,4), followed
by Text (mixed 7bit ASCII, 8bit JIS, and 2x8bit Shift-JIS), lines are
terminated by chr(09h), each page terminated by chr(00h).
```

Demo-Joypad-Data (at P5): ;<-- if none: eight 00h-bytes

```
1 byte - Unknown (usually 05h)
2 byte - Number of following 4-Byte Pairs (N)
N*4 bytes - data (most 4-byte pairs are "xx,FF,FF,FF")
(controller-data for demo/preview, in format: Time,Lsb,Msb,FFh)
(or rather: 8bit time, 12bit joy1, 12bit joy2 ...?)
```

Unused-Joypad-Data (at P6): ;<-- if none: four 00h-bytes

```
Unknown purpose. The GROMs have more controller data here (similar as
at P5), but the existing KROM/ATROM do not seem to use that extra data.
```

Title Bitmap Compression

SNES Decompression Formats

SFC-Box Component List (Cartridges)

SFC-Box Cartridge PCB (GS 0871-102)

```
IC1  28pin  DIP 27C256 EPROM "GROMn-1" (usually 28pin; 28pin/32pin possible)
IC2  20pin  SMD Philips 74HC273D
IC3  20pin  SMD Philips 74HC541D
IC4  20pin  SMD Philips 74HC273D
IC5  14pin  SMD <unknown>
IC6  14pin  SMD <unknown>
IC7  14pin  SMD <unknown>
IC8  16pin  SMD <unknown> 74HC138                    (semi-optional)
IC9  16pin  SMD <unknown> 74HC138                    (semi-optional)
IC10 16pin  SMD <unknown> HC138 or HC130 or so?      (semi-optional)
IC11 16pin  SMD <unknown> 74HC138 or 74HC130 or so ? (semi-optional)
IC12 16pin  SMD <unknown> (near IC16)                (semi-optional)
IC13 16pin  SMD Philips 74HC153D                     (semi-optional)
IC14 20pin  DIP GAL16V8B
IC15 14pin  SMD 74AC125 (near IC17)
IC16 32pin  DIP Sony CXK581000P-12L (SRAM 128Kx8)         (optional)
IC17 28pin  DIP Nintendo DSP1 A/B (for Mario Kart)        (optional)
IC18 14pin  SMD <unknown> 74HC04 (below X1)          (semi-optional)
```

```
  IC19 100pin SMD Mario Chip 1 (Star Fox GSU)               (optional)
  IC20 32pin  SMD SHVC-FO-1    (Star Fox ROM)               (optional)
  IC21 28pin  SMD HY62256A     (Star Fox RAM, 32Kx8)        (optional)
  IC22 14pin  SMD <unknown> (below IC4)
  IC23 14pin  SMD <unknown> HC08                       (semi-optional)
  ROM1 36pin  DIP -or- ROM7  36pin DIP ;\solder pads for up to six ROMs
  ROM2 36pin  DIP -or- ROM8  36pin DIP ; (each with two alternate pin-outs,
  ROM3 36pin  DIP -or- ROM9  36pin DIP ; eg. ROM1=LoROM or ROM7=HiROM)
  ROM4 36pin  DIP -or- ROM10 36pin DIP ; (can be fitted with 32pin/36pin chips
  ROM5 36pin  DIP -or- ROM11 36pin DIP ; except, ROM6 can be 32pin only)
  ROM6 32pin  DIP -or- ROM12 36pin DIP ;/(see IC1 & IC20 for further (EP)ROMs)
  X1   ?pin   DIP <unknown>, oscillator for DSP1, probably 2-3 pins?(optional)
  CN1  100pin DIP OMRON XC5F-0122 Cartridge Connector (female 2x50pin)
```

IC16 is 128K SRAM, this chip is installed in the PSS61 cartridge only, but, it's shared for multiple games (including games in PSS62-PSS64 carts).
The SRAM isn't battery-backed, however, the SFC-Box cannot be switched off (unless when unplugging supply cables), so SRAM should be always receiving a standby-voltage from the console.
The hardware might allow to share the DSP1 chip in similar fashion (?), in the existing carts, it's used only for Mario Kart.
The "optional" components are installed in PSS61 only. The "semi-optional" ones are installed in PSS61 and (for unknown reason) also in PSS62 (whilst PSS63/PSS64 don't have them, although they should be functionally same as PSS62).

Unknown how many different programs are possible (there are pads for max 6 DIP ROMs, plus 1 SMD ROM, but maybe the SMD is alternate to one DIP, and maybe some pads are reserved for games with 2 ROMs; and unknown if the GUI menu supports more than 8 games).

SFC-Box Component List (Console)

SFC-Box Mainboard "MAIN 0871-100A"

```
 Section 1 (CPU/PPU) (Front-Left)
  U1  100pin S-CPU B
  U2  100pin S-PPU1
  U3  100pin S-PPU2 C
  U4  28pin  LH2A256N-10PLL (Mosel-Vitelic, VRAM, 32Kx8)
  U5  28pin  LH2A256N-10PLL (Mosel-Vitelic, VRAM, 32Kx8)
  U6  64pin  S-WRAM A
  U7  24pin  S-ENC A (near APU section)
  U8         N/A ?
  U9  14pin  74HCU04
  U10 8pin   unknown (maybe audio amplifier) (near relay)
  U11 8pin   unknown (maybe audio amplifier) (near relay)
  X1  D21M4 Oscillator (21.47727MHz for S-CPU)
  TC1 Red Trimmer (for above oscillator)
 Section 2 (APU) (Rear-Left)
  IC1 64pin  S-SMP
  IC2 80pin  S-DSP A
  IC3 28pin  HM9453100FP (APU-RAM, 32Kx8)
  IC4 28pin  HM9453100FP (APU-RAM, 32Kx8)
  IC5 16pin  NEC uPD6376 (serial audio D/A converter)
  IC6 8pin   unknown (maybe audio amplifier)
  IC72 28pin MB90082-001 (OSD video controller) (near S-ENC A)
  X2  Blue oscillator (maybe 24.576MHz for APU?)
  X4  D143A4 oscillator (maybe separate NTSC color clock 3.579545MHz mul 4)
  TC2 Trimmer (for X4/D143A4)
  TC3 Trimmer (for IC72/OSD-Chip pin16)
 Section 3 (Rear-Right)
  IC30 80pin  HD64180RF6X (extended Z80 CPU)
  X3  D921B4 oscillator (9.216MHz) (ie. HD64180 clocked at PHI=4.608MHz)
  24 small logic chips (details unknown) (plenty 74HCxxx & 74LSxxx)
 Section 4 (Front-Right)
  18 small logic chips (details unknown)
 Connectors
  CN1 100pin cartridge slot (2x50pin male) (via adaptor to TWO 2x50 slots)
  CN2 44pin  daugtherboard socket (2x22pin male)
  CN3 3pin   unknown/unused (without cable?) (front-right) (coin mechanics?)
  CN4 5pin   Yellow Cable to Front Panel (FR 0871-105) (front-middle)
  CN5 7pin   Yellow Cable to 6-position Keyswitch (front-middle)
  CN6 11pin  Multi-colored Cable (to joypad connectors) (front-left)
  CN? 7pin   Yellow Cable to Modulator (rear-left)
  CN8 6pin   unknown/unused (without cable?) (front-right) (maybe RS232 ???)
      2pin   Black cable to "Nintendo AC Adapter" (Input: DC 5V 10W) (rear)
      2pin   RCA Audio Out Left (rear)
      2pin   RCA Audio Out Right (rear)
 Options & Specials
  SP1 3pin   unknown / jumper (near HD64180) (usually two pins bridged)
  SP2 2pin   unknown / jumper or so (near OSD chip) (usually not bridged)
  SP3 3x5pin unknown / not installed (located in center of mainboard)
  JP1,JP2,JP3,JP4,JP5 - seem to allow to disconnect Shield from GND
  TR1 3pin   unknown (big transistor or so)
  ?   8pin   OMRON G5V-2-5VDC (dual two-position relay) (rear-left)
```

SFC-Box Daughterboard "(unknown PCB name)" (Modulator)
Shielded box with whatever contents, plus external connectors:

```
  2pin   RCA Audio Out Mono      ;\raw A/V (stereo is also available, via
  2pin   RCA Video Out Composite ;/the external connectors on mainboard)
  2pin   RF Out                  ;\
  2pin   ANT In                  ; RF modulated
  ?pin   Channel Select 1CH/2CH  ;/
  7pin   Yellow Cable to Mainboard
```

SFC-Box Daughterboard "Nintendo AC Adapter" (Power Supply)
Remove-able metal box with whatever contents, plus external connectors:

```
  4pin AC OUT 200W MAX (dual 2pin or so)
  2pin DC OUT 5V 5A (via short EXTERNAL cable to Mainboard)
  2pin AC IN (cable to wall socket)
  AC125 5A (Fuse?)
```

SFC-Box Daughterboard "PU 0871-101"

```
  IC1  28pin  DIP 27C512 EPROM "KROM 1" (usually 28pin; 28pin/32pin possible)
```

```
IC2  28pin  SMD SRM20257 (SRAM 32Kx8) (Work-RAM for HD64180, battery-backed)
IC3  16pin  SMD 74HC139
IC4  20pin  SMD 74HC273D (Philips)
IC5  20pin  SMD 74LS541
IC6  16pin  SMD MB3790 (Fujitsu battery controller)
IC7  14pin  SMD S-3520CF (Seiko RTC, Real Time Clock, battery-backed)
IC8  14pin  SMD <unknown>
IC9  14pin  SMD <unknown>
X1   2pin   Oscillator (for RTC) ("S441") (probably 32kHz or so)
TC1         Osc-Adjust (for RTC)
BAT  2pin   Battery (for IC7/RTC and IC2/SRAM)
TM1  8pin   Massive connector (not installed) (maybe FamicomBox-style CATV?)
CN1  44pin  DIP Female Connector 2x22pin (to Mainboard)
```

SFC-Box Daughterboard "GD 0871-103" (Game Cartridge Connectors)

```
CN?  100pin DIP Female Connector 2x22pin (to Mainboard)
CN?  100pin DIP Male Connector 2x22pin (to Cartridge 1)
CN?  100pin DIP Male Connector 2x22pin (to Cartridge 2)
```

SFC-Box Daughterboard "CC 0871-104" (Controller Connectors)

```
11pin  Multicolored Cable to Mainboard
7pin   Controller 1  ;\Standard SNES Joypad connectors (for two standard
7pin   Controller 2  ;/joypads with extra-long cables)
```

SFC-Box Daughterboard "FR 0871-105" (Front Panel)

```
TV-LED and GAME-LED
GAME/TV-Button
RESET-Button
5pin Yellow Cable (to Mainboard)
```

SFC-Box Daughterboard "(unknown PCB name)" (Keyswitch)

```
10-Position Keyswitch (requires a key) (6-positions connected)
7pin Yellow Cable to (to Mainboard) (one common pin, plus 6 switch positions)
```

The 10-position keyswitch is mechanically limited to 6 positions (9,0,1,2,3,4).
There are different keys for different purposes. For example, a "visitor" key can select only the "ON" and "OFF" positions. The right-most position does switch a relay, which does... what? Probably switch-off the SFC-Box?

RTC S-3520 (Real-Time Clock)

Seiko/Epson S-3520CF Serial 4bit Real-Time Clock (RTC)
Contains the usual Time/Date registers, plus 120bit battery-backed RAM (aka 15 bytes) (organized in 2 pages of 15 x 4bits).
This chip is used in both Nintendo Super System (NSS), and in Super Famicom Box.

Seiko/Epson S-3520CF Register Table

```
Index  Bit3   Bit2   Bit1    Bit0  ;Expl.
___Registers in Mode 0_____________ _______________
0      Sec3   Sec2   Sec1    Sec0  ;Seconds, Low     ;\
1      0      Sec6   Sec5    Sec4  ;Seconds, High    ;
2      Min3   Min2   Min1    Min0  ;Minutes, Low     ; Read/Increment-able
3      0      Min6   Min5    Min4  ;Minutes, High    ;
4      Hour3  Hour2  Hour1   Hour0 ;Hours, Low       ; (reading returns the
5      PM/AM  0      Hour5   Hour4 ;Hours, High      ; counter value)
6      0      Week2  Week1   Week0 ;Day of Week      ;
7      Day3   Day2   Day0    Day0  ;Day, Low         ; (writing any dummy
8      0      0      Day5    Day4  ;Day, High        ; value does increment
9      Mon3   Mon2   Mon1    Mon0  ;Month, Low       ; counter value by 1)
A      0      0      0       Mon4  ;Month, High      ;
B      Year3  Year2  Year1   Year0 ;Year, Low        ;
C      Year7  Year6  Year5   Year4 ;Year, High       ;/
D      TPS    30ADJ  CNTR    24/12 ;Control Register ;-Read/Write-able
E      STA    LOST   0       0     ;Status Register  ;-Read only
___Registers in Mode 1_____________ _______________
0-E    x      x      x       x     ;Reserved         ;-Don't use
___Registers in Mode 2_____________ _______________
0-E    SRAM   SRAM   SRAM    SRAM  ;SRAM Page 0      ;-Read/Write-able
___Registers in Mode 3_____________ _______________
0-E    SRAM   SRAM   SRAM    SRAM  ;SRAM Page 1      ;-Read/Write-able
___Mode Register (in Mode 0..3)____ _______________
F      SYSR   TEST   Mode1   Mode0 ;Mode Register    ;-Read/Write-able
```

Whereas, the meaning of the various bits is:

```
Sec    Seconds (BCD, 00h..59h)
Min    Minutes (BCD, 00h..59h)
Hour   Hours   (BCD, 00h..23h or 01h..12h)
Day    Day     (BCD, 01h..31h)
Month  Month   (BCD, 01h..12h)
Year   Year    (BCD, 00h..99h)
Week   Day of Week (0..6) (SFC-Box: Unknown assignment) (NSS: 0=Sunday)
PM/AM  Set for PM, cleared for AM (this is done even when in 24-hour mode)
24/12  24-Hour Mode (0=12, 1=24) (Time/Date may get corrupted when changed?)
TPS    Select Reference Waveform for output on Pin8 (0=1024Hz, 1=1Hz)
30ADJ  Set seconds to zero, and, if seconds was>=30, increase minutes
CNTR   Reset Counters (0=Normal, 1=Reset)
SYSR   Reset Counters and Control/Status/Mode Registers (0=Normal, 1=Reset)
LOST   Time Lost (0=Okay, 1=Lost/Battery failure) (can be reset... how?)
STA    Time Stable (0=Stable/Sec won't change in next 3.9ms, 1=Unstable)
Mode   Mode for Register 0-E (0=RTC, 1=Reserved, 2=SramPage0, 3=SramPage1)
```

If STA=0 then it's safe to read the time (counters won't change within next 3.9ms aka 1/256 seconds). If STA=1 then one should wait until STA=0 before reading the time (else one may miss counter-carry-outs).

Serial Access
Set /CLK and /CS to HIGH as default level. Set /WR to desired direction (before dragging /CS low). Then set /CS to LOW to invoke transfer. Then transfer index/data/garbage (usually 8 clks for WRITES, and 16 clks for READS). Then set /CS back HIGH.
Index/Data/Garbage Nibbles are 4bit each (transferred LSB first). Bits should be output (to DataIn) on falling CLK edge (note: the NSS is doing that properly, the SFC-Box actually outputs data shortly after falling CLK), and can be read (from DataOut) at-or-after raising CLK edge. The separate nibbles are:

```
Nibble  To RTC                    From RTC
```

```
1st     Index I                   Garbage (old index or so)
2nd     Data I    (or dummy)      Garbage (data from old index or so)
3rd     Index II  (or dummy)      Garbage (index I or so)
4th     Data II   (or dummy)      Data I
5th     Index III (or dummy)      Garbage (index II or so)
6th     Data III  (or dummy)      Data II
```

For Writes, one needs to send only 2 nibbles (of which, 2nd nibble is used only for Control & SRAM writes, for Counter-Increment writes it's only a dummy value).
For Reads, one needs to send/receive at least 4 nibbles (though most of them are dummies/garbage; actually used are 1st-To-RTC, and 4th-From-RTC). If desired, one can read two or more registers by reading/writing 6 or more nibbles (the NSS BIOS does so).

Pin-Outs
SNES Pinouts RTC Chips

Z80 CPU Specifications

Z80 Register Set

Register Summary

```
16bit Hi   Lo   Name/Function
--------------------------------------
AF    A    -    Accumulator & Flags
BC    B    C    BC
DE    D    E    DE
HL    H    L    HL
AF'   -    -    Second AF
BC'   -    -    Second BC
DE'   -    -    Second DE
HL'   -    -    Second HL
IX    IXH  IXL  Index register 1
IY    IYH  IYL  Index register 2
SP    -    -    Stack Pointer
PC    -    -    Program Counter/Pointer
-     I    R    Interrupt & Refresh
```

Normal 8bit and 16bit Registers
The Accumulator (A) is the allround register for 8bit operations. Registers B, C, D, E, H, L are normal 8bit registers, which can be also accessed as 16bit register pairs BC, DE, HL.
The HL register pair is used as allround register for 16bit operations. B and BC are sometimes used as counters. DE is used as DEstination pointer in block transfer commands.

Second Register Set
The Z80 includes a second register set (AF',BC',DE',HL') these registers cannot be accessed directly, but can be exchanged with the normal registers by using the EX AF,AF and EXX instructions.

Refresh Register
The lower 7 bits of the Refresh Register (R) are incremented with every instruction. Instructions with at least one prefix-byte (CB,DD,ED,FD, or DDCB,FDCB) will increment the register twice. Bit 7 can be used by programmer to store data. Permanent writing to this register will suppress memory refresh signals, causing Dynamic RAM to lose data.

Interrupt Register
The Interrupt Register (I) is used in interrupt mode 2 only (see command "im 2"). In other modes it can be used as simple 8bit data register.

IX and IY Registers
IX and IY are able to manage almost all the things that HL is able to do. When used as memory pointers they are additionally including a signed index byte (IX+d). The disadvantage is that the opcodes occupy more memory bytes, and that they are less fast than HL-instructions.

Undocumented 8bit Registers
IXH, IXL, IYH, IYL are undocumented 8bit registers which can be used to access high and low bytes of the IX and IY registers (much like H and L for HL). Even though these registers do not officially exist, they seem to be available in all Z80 CPUs, and are quite commonly used by various software.

Z80 Flags

Flag Summary
The Flags are located in the lower eight bits of the AF register pair.

```
Bit Name  Set  Clr  Expl.
0   C     C    NC   Carry Flag
1   N     -    -    Add/Sub-Flag (BCD)
2   P/V   PE   PO   Parity/Overflow-Flag
3   -     -    -    Undocumented
```

```
4   H     -     -    Half-Carry Flag (BCD)
5   -     -     -    Undocumented
6   Z     Z     NZ   Zero-Flag
7   S     M     P    Sign-Flag
```

Carry Flag (C)
This flag signalizes if the result of an arithmetic operation exceeded the maximum range of 8 or 16 bits, ie. the flag is set if the result was less than Zero, or greater than 255 (8bit) or 65535 (16bit). After rotate/shift operations the bit that has been 'shifted out' is stored in the carry flag.

Zero Flag (Z)
Signalizes if the result of an operation has been zero (Z) or not zero (NZ). Note that the flag is set (1) if the result was zero (0).

Sign Flag (S)
Signalizes if the result of an operation is negative (M) or positive (P), the sign flag is just a copy of the most significant bit of the result.

Parity/Overflow Flag (P/V)
This flag is used as Parity Flag, or as Overflow Flag, or for other purposes, depending on the instruction.
Parity: Bit7 XOR Bit6 XOR Bit5 ... XOR Bit0 XOR 1.
8bit Overflow: Indicates if the result was greater/less than +127/-128.
HL Overflow: Indicates if the result was greater/less than +32767/-32768.
After LD A,I or LD A,R: Contains current state of IFF2.
After LDI,LDD,CPI,CPD,CPIR,CPDR: Set if BC<>0 at end of operation.

BCD Flags (H,N)
These bits are solely supposed to be used by the DAA instruction. The N flag signalizes if the previous operation has be an addition or substraction. The H flag indicates if the lower 4 bits exceeded the range from 0-0Fh. (For 16bit instructions: H indicates if the lower 12 bits exceeded the range from 0-0FFFh.)
After adding/subtracting two 8bit BCD values (0-99h) the DAA instruction can be used to convert the hexadecimal result in the A register (0-FFh) back to BCD format (0-99h). Note that DAA also requires the carry flag to be set correctly, and thus should not be used after INC A or DEC A.

Undocumented Flags (Bit 3,5)
The content of these undocumented bits is filled by garbage by all instructions that affect one or more of the normal flags (for more info read the chapter Garbage in Flag Register), the only way to read out these flags would be to copy the flags register onto the stack by using the PUSH AF instruction.
However, the existence of these bits makes the AF register a full 16bit register, so that for example the code sequence PUSH DE, POP AF, PUSH AF, POP HL would set HL=DE with all 16bits intact.

Z80 Instruction Format

Commands and Parameters
Each instruction consists of a command, and optionally one or two parameters. Usually the leftmost parameter is modified by the operation when two parameters are specified.

Parameter Placeholders
The following placeholders are used in the following chapters:

```
r      8bit  register A,B,C,D,E,H,L
rr     16bit register BC, DE, HL/IX/IY, AF/SP   (as described)
i      8bit  register A,B,C,D,E,IXH/IYH,IXL/IYL
ii     16bit register IX,IY
n      8bit  immediate 00-FFh                   (unless described else)
nn     16bit immediate 0000-FFFFh
d      8bit  signed offset -128..+127
f      flag  condition nz,z,nc,c AND/OR po,pe,p,m  (as described)
(..)   16bit pointer to byte/word in memory
```

Opcode Bytes
Each command (including parameters) consists of 1-4 bytes. The respective bytes are described in the following chapters. In some cases the register number or other parameters are encoded into some bits of the opcode, in that case the opcode is specified as "xx". Opcode prefix bytes "DD" (IX) and "FD" (IY) are abbreviated as "pD".

Clock Cycles
The clock cycle values in the following chapters specify the execution time of the instruction. For example, an 8-cycle instruction would take 2 microseconds on a CPU which is operated at 4MHz (8/4 ms). For conditional instructions two values are specified, for example, 17;10 means 17 cycles if condition true, and 10 cycles if false.
Note that in case that WAIT signals are sent to the CPU by the hardware then the execution may take longer.

Affected Flags
The instruction tables below are including a six character wide field for the six flags: Sign, Zero, Halfcarry, Parity/Overflow, N-Flag, and Carry (in that order).
The meaning of the separate characters is:

```
s    Indicates Signed result
z    Indicates Zero
h    Indicates Halfcarry
o    Indicates Overflow
p    Indicates Parity
c    Indicates Carry
-    Flag is not affected
0    Flag is cleared
1    Flag is set
x    Flag is destroyed (unspecified)
i    State of IFF2
e    Indicates BC<>0 for LDX(R) and CPX(R), or B=0 for INX(R) and OUTX(R)
```

Z80 Load Commands

8bit Load Commands

```
Instruction    Opcode  Cycles Flags  Notes
ld   r,r       xx           4 ------ r=r
ld   i,i       pD xx        8 ------ i=i
ld   r,n       xx nn        7 ------ r=n
```

```
ld   i,n       pD xx nn    11 ------ i=n
ld   r,(HL)    xx           7 ------ r=(HL)
ld   r,(ii+d)  pD xx dd    19 ------ r=(ii+d)
ld   (HL),r    7x           7 ------ (HL)=r
ld   (ii+d),r  pD 7x dd    19 ------
ld   (HL),n    36 nn       10 ------
ld   (ii+d),n  pD 36 dd nn 19 ------
ld   A,(BC)    0A           7 ------
ld   A,(DE)    1A           7 ------
ld   A,(nn)    3A nn nn    13 ------
ld   (BC),A    02           7 ------
ld   (DE),A    12           7 ------
ld   (nn),A    32 nn nn    13 ------
ld   A,I       ED 57        9 sz0i0- A=I  ;Interrupt Register
ld   A,R       ED 5F        9 sz0i0- A=R  ;Refresh Register
ld   I,A       ED 47        9 ------
ld   R,A       ED 4F        9 ------
```

16bit Load Commands

```
Instruction    Opcode  Cycles Flags  Notes
ld   rr,nn     x1 nn nn    10 ------ rr=nn    ;rr may be BC,DE,HL or SP
ld   ii,nn     pD 21 nn nn 13 ------ ii=nn
ld   HL,(nn)   2A nn nn    16 ------ HL=(nn)
ld   ii,(nn)   pD 2A nn nn 20 ------ ii=(nn)
ld   rr,(nn)   ED xB nn nn 20 ------ rr=(nn)  ;rr may be BC,DE,HL or SP
ld   (nn),HL   22 nn nn    16 ------ (nn)=HL
ld   (nn),ii   pD 22 nn nn 20 ------ (nn)=ii
ld   (nn),rr   ED x3 nn nn 20 ------ (nn)=rr  ;rr may be BC,DE,HL or SP
ld   SP,HL     F9           6 ------ SP=HL
ld   SP,ii     pD F9       10 ------ SP=ii
push rr        x5          11 ------ SP=SP-2, (SP)=rr  ;rr may be BC,DE,HL,AF
push ii        pD E5       15 ------ SP=SP-2, (SP)=ii
pop  rr        x1          10 (-AF-) rr=(SP), SP=SP+2  ;rr may be BC,DE,HL,AF
pop  ii        pD E1       14 ------ ii=(SP), SP=SP+2
ex   DE,HL     EB           4 ------ exchange DE <--> HL
ex   AF,AF     08           4 xxxxxx exchange AF <--> AF'
exx            D9           4 ------ exchange BC,DE,HL <--> BC',DE',HL'
ex   (SP),HL   E3          19 ------ exchange (SP) <--> HL
ex   (SP),ii   pD E3       23 ------ exchange (SP) <--> ii
```

Blocktransfer

```
Instruction    Opcode  Cycles Flags  Notes
ldi            ED A0       16 --0e0- (DE)=(HL), HL=HL+1, DE=DE+1, BC=BC-1
ldd            ED A8       16 --0e0- (DE)=(HL), HL=HL-1, DE=DE-1, BC=BC-1
ldir           ED B0  bc*21-5 --0?0- ldi-repeat until BC=0
lddr           ED B8  bc*21-5 --0?0- ldd-repeat until BC=0
```

Z80 Arithmetic/Logical Commands

8bit Arithmetic/Logical Commands

```
Instruction    Opcode  Cycles Flags  Notes
daa            27           4 szxp-x decimal adjust akku
cpl            2F           4 --1-1- A = A xor FF
neg            ED 44        8 szho1c A = 00-A
<arit>  r      xx           4 szhonc see below
<arit>  i      pD xx        8 szhonc see below, UNDOCUMENTED
<arit>  n      xx nn        7 szhonc see below
<arit>  (HL)   xx           7 szhonc see below
<arit>  (ii+d) pD xx dd    19 szhonc see below
<cnt>   r      xx           4 szhon- see below
<cnt>   i      pD xx        8 szhon- see below, UNDOCUMENTED
<cnt>   (HL)   xx          11 szhon- see below
<cnt>   (ii+d) pD xx dd    23 szhon- see below
<logi>  r      xx           4 szhp00 see below
<logi>  i      pD xx        8 szhp00 see below, UNDOCUMENTED
<logi>  n      xx nn        7 szhp00 see below
<logi>  (HL)   xx           7 szhp00 see below
<logi>  (ii+d) pD xx dd    19 szhp00 see below
```

Arithmetic <arit> commands:

```
add   A,op     see above 4-19 szho0c A=A+op
adc   A,op     see above 4-19 szho0c A=A+op+cy
sub   op       see above 4-19 szho1c A=A-op
sbc   A,op     see above 4-19 szho1c A=A-op-cy
cp    op       see above 4-19 szho1c compare, ie. VOID=A-op
```

Increment/Decrement <cnt> commands:

```
inc   op       see above 4-23 szho0- op=op+1
dec   op       see above 4-23 szho1- op=op-1
```

Logical <logi> commands:

```
and   op       see above 4-19 sz1p00 A=A & op
xor   op       see above 4-19 sz0p00 A=A XOR op
or    op       see above 4-19 sz0p00 A=A | op
```

16bit Arithmetic Commands

```
Instruction    Opcode  Cycles Flags  Notes
add  HL,rr     x9          11 --h-0c HL = HL+rr    ;rr may be BC,DE,HL,SP
add  ii,rr     pD x9       15 --h-0c ii = ii+rr    ;rr may be BC,DE,ii,SP (!)
adc  HL,rr     ED xA       15 szho0c HL = HL+rr+cy ;rr may be BC,DE,HL,SP
sbc  HL,rr     ED x2       15 szho1c HL = HL-rr-cy ;rr may be BC,DE,HL,SP
inc  rr        x3           6 ------ rr = rr+1     ;rr may be BC,DE,HL,SP
inc  ii        pD 23       10 ------ ii = ii+1
dec  rr        xB           6 ------ rr = rr-1     ;rr may be BC,DE,HL,SP
dec  ii        pD 2B       10 ------ ii = ii-1
```

Searchcommands

```
Instruction    Opcode  Cycles Flags  Notes
cpi            ED A1       16 szhe1- compare A-(HL), HL=HL+1, DE=DE+1, BC=BC-1
cpd            ED A9       16 szhe1- compare A-(HL), HL=HL-1, DE=DE-1, BC=BC-1
```

```
cpdr          ED B9   x*21-5 szhe1- cpd-repeat until BC=0 or compare fits
```

Z80 Rotate/Shift and Singlebit Operations

Rotate and Shift Commands

```
Instruction    Opcode  Cycles Flags  Notes
rlca           07           4 --0-0c rotate akku left
rla            17           4 --0-0c rotate akku left through carry
rrca           0F           4 --0-0c rotate akku right
rra            1F           4 --0-0c rotate akku right through carry
rld            ED 6F       18 sz0p0- rotate left low digit of A through (HL)
rrd            ED 67       18 sz0p0- rotate right low digit of A through (HL)
<cmd> r        CB xx        8 sz0p0c see below
<cmd> (HL)     CB xx       15 sz0p0c see below
<cmd> (ii+d)   pD CB dd xx 23 sz0p0c see below
<cmd> r,(ii+d) pD CB dd xx 23 sz0p0c see below, UNDOCUMENTED modify and load
```

Whereas <cmd> may be:

```
rlc    rotate left
rl     rotate left through carry
rrc    rotate right
rr     rotate right through carry
sla    shift left arithmetic (b0=0)
sll    UNDOCUMENTED shift left (b0=1)
sra    shift right arithmetic (b7=b7)
srl    shift right logical (b7=0)
```

Singlebit Operations

```
Instruction    Opcode  Cycles Flags  Notes
bit  n,r       CB xx        8 xz1x0- test bit n  ;n=0..7
bit  n,(HL)    CB xx       12 xz1x0-
bit  n,(ii+d)  pD CB dd xx 20 xz1x0-
set  n,r       CB xx        8 ------ set bit n   ;n=0..7
set  n,(HL)    CB xx       15 ------
set  n,(ii+d)  pD CB dd xx 23 ------
set r,n,(ii+d) pD CB dd xx 23 ------ UNDOCUMENTED set n,(ii+d) and ld r,(ii+d)
res  n,r       CB xx        8 ------ reset bit n ;n=0..7
res  n,(HL)    CB xx       15 ------
res  n,(ii+d)  pD CB dd xx 23 ------
res r,n,(ii+d) pD CB dd xx 23 ------ UNDOCUMENTED res n,(ii+d) and ld r,(ii+d)
ccf            3F           4 --h-0c h=cy, cy=cy xor 1
scf            37           4 --0-01 cy=1
```

Z80 Jumpcommands & Interrupts

General Jump Commands

```
Instruction    Opcode  Cycles Flags  Notes
jp   nn        C3 nn nn    10 ------ jump to nn, ie. PC=nn
jp   HL        E9           4 ------ jump to HL, ie. PC=HL
jp   ii        pD E9        8 ------ jump to ii, ie. PC=ii
jp   f,nn      xx nn nn 10;10 ------ jump to nn if nz,z,nc,c,po,pe,p,m
jr   nn        18 dd       12 ------ relative jump to nn, ie. PC=PC+d
jr   f,nn      xx dd     12;7 ------ relative jump to nn if nz,z,nc,c
djnz nn        10 dd     13;8 ------ B=B-1 and relative jump to nn if B<>0
call nn        CD nn nn    17 ------ call nn ie. SP=SP-2, (SP)=PC, PC=nn
call f,nn      xx nn nn 17;10 ------ call nn if nz,z,nc,c,po,pe,p,m
ret            C9          10 ------ pop PC ie. PC=(SP), SP=SP+2
ret  f         xx        11;5 ------ pop PC if nz,z,nc,c,po,pe,p,m
rst  n         xx          11 ------ call n  ;n=00,08,10,18,20,28,30,38
nop            00           4 ------ no operation
```

Interrupt Related Commands

```
Instruction    Opcode  Cycles Flags  Notes
di             F3           4 ------ IFF1=0, IFF2=0  ;disable interrupts
ei             FB           4 ------ IFF1=1, IFF2=1  ;enable interrupts
im   0         ED 46        8 ------ read opcode from databus on interrupt
im   1         ED 56        8 ------ execute call 0038h on interrupt
im   2         ED 5E        8 ------ execute call (i*100h+databus) on int.
halt           76         N*4 ------ repeat until interrupt occurs
reti           ED 4D       14 ------ pop PC, IFF1=IFF2, ACK (ret from INT)
retn           ED 45       14 ------ pop PC, IFF1=IFF2      (ret from NMI)
</INT=LOW,IM=0,IFF1=1>  1+var ------ IFF1=0,IFF2=0, exec opcode from databus
</INT=LOW,IM=1,IFF1=1>     12 ------ IFF1=0,IFF2=0, CALL 0038h
</INT=LOW,IM=2,IFF1=1>     18 ------ IFF1=0,IFF2=0, CALL [I*100h+databus]
</NMI=falling_edge>         ? ------ IFF1=0,        CALL 0066h
```

Z80 I/O Commands

```
Instruction    Opcode  Cycles Flags  Notes
in   A,(n)     DB nn       11 ------ A=PORT(A*100h+n)
in   r,(C)     ED xx       12 sz0p0- r=PORT(BC)
in   (C)       ED 70       12 sz0p0- **undoc/illegal** VOID=PORT(BC)
out  (n),A     D3 nn       11 ------ PORT(A*100h+n)=A
out  (C),r     ED xx       12 ------ PORT(BC)=r
out  (C),0     ED 71       12 ------ **undoc/illegal** PORT(BC)=00
ini            ED A2       16 xexxxx MEM(HL)=PORT(BC), HL=HL+1, B=B-1
ind            ED AA       16 xexxxx MEM(HL)=PORT(BC), HL=HL-1, B=B-1
outi           ED A3       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL+1
outd           ED AB       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL-1
inir           ED B2   b*21-5 x1xxxx same than ini, repeat until b=0
indr           ED BA   b*21-5 x1xxxx same than ind, repeat until b=0
otir           ED B3   b*21-5 x1xxxx same than outi, repeat until b=0
otdr           ED BB   b*21-5 x1xxxx same than outd, repeat until b=0
```

Z80 Interrupts

Interrupt Flip-Flop (IFF1,IFF2)
The IFF1 flag is used to enable/disable INTs (maskable interrupts).
In a raw INT-based system, IFF2 is always having the same state than IFF1. However, in a NMI-based system the IFF2 flag is used to backup the recent IFF1 state prior to NMI execution, and may be used to restore IFF1 upon NMI completion by RETN opcode.
Beside for the above 'backup' function, IFF2 itself is having no effect. Neither IFF1 nor IFF2 affect NMIs which are always enabled.
The following opcodes/events are modifying IFF1 and/or IFF2:

```
EI     IFF1=1, IFF2=1
DI     IFF1=0, IFF2=0
<INT>  IFF1=0, IFF2=0
<NMI>  IFF1=0
RETI   IFF1=IFF2
RETN   IFF1=IFF2
```

When using the EI instruction, the new IFF state isn't applied until the next instruction has completed (this ensures that an interrupt handler which is using the sequence "EI, RET" may return to the main program before the next interrupt is executed).
Interrupts can be disabled by the DI instruction (IFF=0), and are additionally automatically each time when an interrupt is executed.

Interrupt Execution
An interrupt is executed when an interrupt is requested by the hardware, and IFF is set. Whenever both conditions are true, the interrupt is executed after the completion of the current opcode.
Note that repeated block commands (such like LDIR) can be interrupted also, the interrupt return address on the stack then points to the interrupted opcode, so that the instruction may continue as normal once the interrupt handler returns.

Interrupt Modes (IM 0,1,2)
The Z80 supports three interrupt modes which can be selected by IM 0, IM 1, and IM 2 instructions. The table below describes the respective operation and execution time in each mode.

```
Mode  Cycles  Refresh  Operation
0     1+var   0+var    IFF1=0,IFF2=0, read and execute opcode from databus
1     12      1        IFF1=0,IFF2=0, CALL 0038h
2     18      1        IFF1=0,IFF2=0, CALL [I*100h+databus]
```

Mode 0 requires an opcode to be output to the databus by external hardware, in case that no byte is output, and provided that the 'empty' databus is free of garbage, then the CPU might tend to read a value of FFh (opcode RST 38h, 11 cycles, 1 refresh) - the clock cycles (11+1), refresh cycles (1), and executed operation are then fully identical as in Mode 1.
Mode 1 interrupts always perform a CALL 0038h operation. The downside is that many systems may have ROM located at this address, making it impossible to hook the interrupt handler directly.

Mode 2 calls to a 16bit address which is read from a table in memory, the table pointer is calculated from the "I" register (initialized by LD I,A instruction) multiplied by 100h, plus an index byte which is read from the databus. The following trick may be used to gain stable results in Mode 2 even if no index byte is supplied on the databus: For example, set I=40h the origin of the table will be then at 4000h in memory. Now fill the entire area from 4000h to 4100h (101h bytes, including 4100h) by the value 41h. The CPU will then perform a CALL 4141h upon interrupt execution - regardless of whether the randomized index byte is an even or odd number.

Non-Maskable Interrupts (NMIs)
Unlike INTs, NMIs cannot be disabled by the CPU, ie. DI and EI instructions and the state of IFF1 and IFF2 do not have effect on NMIs. The NMI handler address is fixed at 0066h, regardless of the interrupt mode (IM). Upon NMI execution, IFF1 is cleared (disabeling maskable INTs - NMIs remain enabled, which may result in nested execution if the handler does not return before next NMI is requested). IFF2 remains unchanged, thus containing the most recent state of IFF1, which may be used to restore IFF1 if the NMI handler returns by RETN instruction.
Execution time for NMIs is unknown (?).

RETN (return from NMI and restore IFF1)
Intended to return from NMI and to restore the old IFF1 state (assuming the old state was IFF1/IFF2 both set or both cleared).

RETI (return from INT with external acknowledge)
Intended to return from INT and to notify peripherals about completion of the INT handler, the Z80 itself doesn't send any such acknowledge signal (instead, peripherals like Z80-PIO or Z80-SIO must decode the databus during /M1 cycles, and identify the opcode sequence EDh,4Fh as RETI). Aside from such external handling, internally, RETI is exactly same as RETN, and, like RETN it does set IFF1=IFF2 (though in case of RETI this is a dirt effect without practical use; within INT handlers IFF1 and IFF2 are always both zero, or when EI was used both set). Recommended methods to return from INT are: EI+RETI (when needing the external acknowledge), or EI+RET (faster).

Z80 Meaningless and Duplicated Opcodes

Mirrored Instructions
NEG (ED44) is mirrored to ED4C,54,5C,64,6C,74,7C.
RETN (ED45) is mirrored to ED55,65,75.
RETI (ED4D) is mirrored to ED5D,6D,7D.

Mirrored IM Instructions
IM 0,X,1,2 (ED46,4E,56,5E) are mirrored to ED66,6E,76,7E.
Whereas IM X is an undocumented mirrored instruction itself which appears to be identical to either IM 0 or IM 1 instruction (?).

Duplicated LD HL Instructions
LD (nn),HL (opcode 22NNNN) is mirrored to ED63NNNN.
LD HL,(nn) (opcode 2ANNNN) is mirrored to ED6BNNNN.
Unlike the other instructions in this chapter, these two opcodes are officially documented. The clock/refresh cycles for the mirrored instructions are then 20/2 instead of 16/1 as for the native 8080 instructions.

Mirrored BIT N,(ii+d) Instructions
Unlike as for RES and SET, the BIT instruction does not support a third operand, ie. DD or FD prefixes cannot be used on a BIT N,r instruction in order to produce a BIT r,N,(ii+d) instruction. When attempting this, the 'r' operand is ignored, and the resulting instruction is identical to BIT N,(ii+d).
Except that, not tested yet, maybe undocumented flags are then read from 'r' instead of from ii+d(?).

Non-Functional Opcodes
The following opcodes behave much like the NOP instruction

ED00-3F, ED77, ED7F, ED80-9F, EDA4-A7, EDAC-AF, EDB4-B7, EDBC-BF, EDC0-FF.
The execution time for these opcodes is 8 clock cycles, 2 refresh cycles.
Note that some of these opcodes appear to be used for additional instructions by the R800 CPU in newer turbo R (MSX) models.

Ignored DD and FD Prefixes
In some cases, DD-prefixes (IX) and FD-prefixes (IY) may be ignored by the CPU. This happens when using one (or more) of the above prefixes prior to instructions that already contain an ED, DD, or FD prefix, or prior to any instructions that do not support IX, IY, IXL, IXH, IYL, IYH operands. In such cases, 4 clock cycles and 1 refresh cycle are counted for each ignored prefix byte.

Z80 Garbage in Flag Register

Nocash Z80-flags description
This chapter describes the undocumented Z80 flags (bit 3 and 5 of the Flags Register), these flags are affected by ALL instructions that modify one or more of the normal flags - all OTHER instructions do NOT affect the undocumented flags.

For some instructions, the content of some flags has been officially documented as 'destroyed', indicating that the flags contain garbage, the exact garbage calculation for these instructions will be described here also.

All information below just for curiosity. Keep in mind that Z80 compatible CPUs (or emulators) may not supply identical results, so that it wouldn't be a good idea to use these flags in any programs (not that they could be very useful anyways).

Normal Behaviour for Undocumented Flags
In most cases, undocumented flags are copied from the Bit 3 and Bit 5 of the result byte. That is "A AND 28h" for:
```
RLD; CPL; RLCA; RLA; LD A,I; ADD OP; ADC OP; XOR OP; AND OP;
RRD; NEG; RRCA; RRA; LD A,R; SUB OP; SBC OP; OR OP ; DAA.
```
When other operands than A may be modified, "OP AND 28h" for:
```
RLC OP; RL OP; SLA OP; SLL OP; INC OP; IN OP,(C);
RRC OP; RR OP; SRA OP; SRL OP; DEC OP
```
For 16bit instructions flags are calculated as "RR AND 2800h":
```
ADD RR,XX; ADC RR,XX; SBC RR,XX.
```

Slightly Special Undocumented Flags
For 'CP OP' flags are calculated as "OP AND 28h", that is the unmodified operand, and NOT the internally calculated result of the comparision.
For 'SCF' and 'CCF' flags are calculated as "(A OR F) AND 28h", ie. the flags remain set if they have been previously set.
For 'BIT N,R' flags are calculated as "OP AND 28h", additionally the P-Flag is set to the same value than the Z-Flag (ie. the Parity of "OP AND MASK"), and the S-flag is set to "OP AND MASK AND 80h".

Fatal MEMPTR Undocumented Flags
For 'BIT N,(HL)' the P- and S-flags are set as for BIT N,R, but the undocumented flags are calculated as "MEMPTR AND 2800h", for more info about MEMPTR read on below.
The same applies to 'BIT N,(ii+d)', but the result is less unpredictable because the instruction sets MEMPTR=ii+d, so that undocumented flags are "<ii+d> AND 2800h".

Memory Block Command Undocumented Flags
For LDI, LDD, LDIR, LDDR, undocumented flags are "((A+DATA) AND 08h) + ((A+DATA) AND 02h)*10h".
For CPI, CPD, CPIR, CPDR, undocumented flags are "((A-DATA-FLG_H) AND 08h) + ((A-DATA-FLG_H) AND 02h)*10h", whereas the CPU first calculates A-DATA, and then internally subtracts the resulting H-flag from the result.

Chaotic I/O Block Command Flags
The INI, IND, INIR, INDR, OUTI, OUTD, OTIR, OTDR instructions are doing a lot of obscure things, to simplify the description a placeholder called DUMMY is used in the formulas.
```
DUMMY = "REG_C+DATA+1"    ;for INI/INIR
DUMMY = "REG_C+DATA-1"    ;for IND/INDR
DUMMY = "REG_L+DATA"      ;for OUTI,OUTD,OTIR,OTDR
FLG_C = Carry  of above "DUMMY" calculation
FLG_H = Carry  of above "DUMMY" calculation (same as FLG_C)
FLG_N = Sign   of "DATA"
FLG_P = Parity of "REG_B XOR (DUMMY AND 07h)"
FLG_S = Sign   of "REG_B"
UNDOC = Bit3,5 of "REG_B AND 28h"
```
The above registers L and B are meant to contain the new values which are already incremented/decremented by the instruction.
Note that the official docs mis-described the N-Flag as set, and the C-Flag as not affected.

DAA Flags
Addition (if N was 0):
```
FLG_H = (OLD_A AND 0Fh) > 09h
FLG_C = Carry of result
```
Subtraction (if N was 1):
```
FLG_H = (NEW_A AND 0Fh) > 09h
FLG_C = OLD_CARRY OR (OLD_A>99h)
```
For both addition and subtraction, N remains unmodified, and S, Z, P contain "Sign", Zero, and Parity of result (A). Undocumented flags are set to (A AND 28h) as normal.

Mis-documented Flags
For all XOR/OR: H=N=C=0, and for all AND: H=1, N=C=0, unlike described else in Z80 docs. Also note C,N flag description bug for I/O block commands (see above).

Internal MEMPTR Register
This is an internal Z80 register, modified by some instructions, and usually completely hidden to the user, except that Bit 11 and Bit 13 can be read out at a later time by BIT N,(HL) instructions.
The following list specifies the resulting content of the MEMPTR register caused by the respective instructions.
```
Content Instruction
A*100h  LD (xx),A               ;xx=BC,DE,nn
xx+1    LD A,(xx)               ;xx=BC,DE,nn
nn+1    LD (nn),rr; LD rr,(nn)  ;rr=BC,DE,HL,IX,IY
rr      EX (SP),rr              ;rr=HL,IX,IY (MEMPTR=new value of rr)
rr+1    ADD/ADC/SBC rr,xx       ;rr=HL,IX,IY (MEMPTR=old value of rr+1)
HL+1    RLD and RRD
```

```
dest    JP nn; CALL nn; JR nn   ;dest=nn
dest    JP f,nn; CALL f,nn      ;regardless of condition true/false
dest    RET; RETI; RETN         ;dest=value read from (sp)
dest    RET f; JR f,nn; DJNZ nn ;only if condition=true
00XX    RST n
adr+1   IN A,(n)                ;adr=A*100h+n, memptr=A*100h+n+1
bc+1    IN r,(BC); OUT (BC),r   ;adr=bc
ii+d    All instructions with operand (ii+d)
```

Also the following might or might not affect MEMPTR, not tested yet:

```
OUT (N),A and block commands LDXX, CPXX, INXX, OUTXX
and probably interrupts in IM 0, 1, 2
```

All other commands do not affect the MEMPTR register - this includes all instructions with operand (HL), all PUSH and POP instructions, not executed conditionals JR f,d, DJNZ d, RET f (ie. with condition=false), and the JP HL/IX/IY jump instructions.

Z80 Compatibility

The Z80 CPU is (almost) fully backwards compatible to older 8080 and 8085 CPUs.

Instruction Format
The Z80 syntax simplifies the chaotic 8080/8085 syntax. For example, Z80 uses the command "LD" for all load instructions, 8080/8085 used various different commands depending on whether the operands are 8bit registers, 16bit registers, memory pointers, and/or an immediates. However, these changes apply to the source code only - the generated binary code is identical for both CPUs.

Parity/Overflow Flag
The Z80 CPU uses Bit 2 of the flag register as Overflow flag for arithmetic instructions, and as Parity flag for other instructions. 8080/8085 CPUs are always using this bit as Parity flag for both arithmetic and non-arithmetic instructions.

Z80 Specific Instructions
The following instructions are available for Z80 CPUs only, but not for older 8080/8085 CPUs:
All CB-prefixed opcodes (most Shift/Rotate, all BIT/SET/RES commands).
All ED-prefixed opcodes (various instructions, and all block commands).
All DD/FD-prefixed opcodes (registers IX and IY).
As well as DJNZ nn; JR nn; JR f,nn; EX AF,AF; and EXX.

8085 Specific Instructions
The 8085 instruction set includes two specific opcodes in addition to the 8080 instruction set, used to control 8085-specifc interrupts and SID and SOD input/output signals. These opcodes, RIM (20h) and SIM (30h), are not supported by Z80/8080 CPUs.

Z80 vs Z80A
Both Z80 and Z80A are including the same instruction set, the only difference is the supported clock frequency (Z80 = max 2.5MHz, Z80A = max 4MHz).

NEC-780 vs Zilog-Z80
These CPUs are apparently fully compatible to each other, including for undocumented flags and undocumented opcodes.

Z80 Pin-Outs

```
      ____  ____
     |    |_|    |
 A11 |1         40| A10
 A12 |2         39| A9
 A13 |3         38| A8
 A14 |4         37| A7
 A15 |5         36| A6
 CLK |6         35| A5
  D4 |7         34| A4
  D3 |8         33| A3
  D5 |9         32| A2
  D6 |10  Z80  31| A1
 VCC |11  CPU  30| A0
  D2 |12        29| GND
  D7 |13        28| /RFSH
  D0 |14        27| /M1
  D1 |15        26| /RST
/INT |16        25| /BUSRQ
/NMI |17        24| /WAIT
/HALT|18        23| /BUSAK
/MREQ|19        22| /WR
/IORQ|20        21| /RD
     |___________|
```

Z80 Local Usage

Nintendo Super System (Z80)
Clocked at 4.000MHz.
NMIs are used for something (probably Vblank or Vsync or so). Normal interrupts seem to be unused. There is MAYBE no watchdog hardware (but the BIOS is using a software-based watchdog; namely, it's misusing the "I" register as watchdog timer; decreased by NMI handler). ALTHOUGH, like the PC10, it might ADDITIONALLY have a hardware watchdog...?

Super Famicom Box (HD64180)
Clocked at by a 9.216MHz oscillator, ie. the HD64180 is internally clocked at PHI=4.608MHz.

HD64180

The HD64180/Z180 are enhanced Z80 CPUs. The HD64180 was originally made by Hitachi, and later enhanced to Z180 by Zilog.

The system clock (PHI) is half the frequency of the crystal. Supported PHI values are 6.144MHz, 4.608MHz, 3.072MHz (these values allow to program ASCI channels to valid RS232 baudrates).

HD64180 Internal I/O Map

HD64180 Internal Registers
Internal I/O Ports are initially mapped to Port 0000h..003Fh (but can be reassigned to 0040h..007Fh, 0080h..00BFh, or 00C0h..00FFh via ICR register).

```
Port Name     Expl.                                  (On Reset)
00h  CNTLA0   ASCI Channel 0 Control Reg A           (10h, bit3=var)
01h  CNTLA1   ASCI Channel 1 Control Reg A           (10h, bit3/bit4=var)
02h  CNTLB0   ASCI Channel 0 Control Reg B           (07h, bit7/bit5=var)
03h  CNTLB1   ASCI Channel 1 Control Reg B           (07h, bit7=var)
04h  STAT0    ASCI Channel 0 Status Register         (00h, bit2/bit1=var)
05h  STAT1    ASCI Channel 1 Status Register         (02h)
06h  TDR0     ASCI Channel 0 Transmit Data Register
07h  TDR1     ASCI Channel 1 Transmit Data Register
08h  RDR0     ASCI Channel 0 Receive Data Register
09h  RDR1     ASCI Channel 1 Receive Data Register
0Ah  CNTR     CSI/O Control Register                 (0Fh)
0Bh  TRDR     CSI/O Transmit/Receive Data Register
0Ch  TMDR0L   Timer 0 Counter "Data" Register, Bit0-7  (FFh)
0Dh  TMDR0H   Timer 0 Counter "Data" Register, Bit8-15 (FFh)
0Eh  RLDR0L   Timer 0 Reload Register, Bit0-7        (FFh)
0Fh  RLDR0H   Timer 0 Reload Register, Bit8-15       (FFh)
10h  TCR      Timer Control Register                 (00h)
11h-13h       Reserved
 12h  ASEXT0   ASCI Channel 0 Extension Control Reg ;\Z8S180/Z8L180 only
 13h  ASEXT1   ASCI Channel 0 Extension Control Reg ;/(not Z80180/HD64180)
14h  TMDR1L   Timer 1 Counter "Data" Register, Bit0-7  (FFh)
15h  TMDR1H   Timer 1 Counter "Data" Register, Bit8-15 (FFh)
16h  RLDR1L   Timer 1 Reload Register, Bit0-7        (FFh)
17h  RLDR1H   Timer 1 Reload Register, Bit8-15       (FFh)
18h  FRC      Free Running Counter                   (FFh)
19h-1Fh       Reserved
 1Ah  ASTC0L   ASCI Channel 0 Time Constant, Bit0-7  ;\
 1Bh  ASTC0H   ASCI Channel 0 Time Constant, Bit8-15 ; Z8S180/Z8L180 only
 1Ch  ASTC1L   ASCI Channel 1 Time Constant, Bit0-7  ; (not Z80180/HD64180)
 1Dh  ASTC1H   ASCI Channel 1 Time Constant, Bit8-15 ;
 1Eh  CMR      Clock Multiplier Register            ;
 1Fh  CCR      CPU Control Register                 ;/
20h  SAR0L    DMA Channel 0 Source Address, Bit0-7 (Memory or I/O)
21h  SAR0H    DMA Channel 0 Source Address, Bit8-15 (Memory or I/O)
22h  SAR0B    DMA Channel 0 Source Address, Bit16-19 (Memory or DRQ)
23h  DAR0L    DMA Channel 0 Destination Address, Bit0-7 (Memory or I/O)
24h  DAR0H    DMA Channel 0 Destination Address, Bit8-15 (Memory or I/O)
25h  DAR0B    DMA Channel 0 Destination Address, Bit16-19 (Memory or DRQ)
26h  BCR0L    DMA Channel 0 Byte Count Register, Bit0-7
27h  BCR0H    DMA Channel 0 Byte Count Register, Bit8-15
28h  MAR1L    DMA Channel 1 Memory Address, Bit0-7 (Source or Dest)
29h  MAR1H    DMA Channel 1 Memory Address, Bit8-15 (Source or Dest)
2Ah  MAR1B    DMA Channel 1 Memory Address, Bit16-19 (Source or Dest)
2Bh  IAR1L    DMA Channel 1 I/O Address, Bit0-7 (Dest or Source)
2Ch  IAR1H    DMA Channel 1 I/O Address, Bit8-15 (Dest or Source)
 2Dh           Reserved ;IAR1B on Z8S180/Z8L180 (not Z80180/HD64180)
2Eh  BCR1L    DMA Channel 1 Byte Count Register, Bit0-7
2Fh  BCR1H    DMA Channel 1 Byte Count Register, Bit8-15
30h  DSTAT    DMA "Status" Register                  (32h on Reset)
31h  DMODE    DMA Mode Register                      (C1h on Reset)
32h  DCNTL    DMA/WAIT Control Register              (F0h on Reset)
33h  IL       Interrupt Vector Low Register          (00h on Reset)
34h  ITC      INT/TRAP Control Register              (39h on Reset)
35h           Reserved
36h  RCR      Refresh Control Register               (FCh on Reset)
37h           Reserved
38h  CBR      MMU Common Base Register (Common Area 1)(00h on Reset)
39h  BBR      MMU Bank Base Register (Bank Area)     (00h on Reset)
3Ah  CBAR     MMU Common/Bank Area Register          (F0h on Reset)
3Bh-3Dh       Reserved
3Eh  OMCR     Operation Mode, Z180 only (not HD64180) (FFh on Reset)
3Fh  ICR      I/O Control Register                   (1Fh on Reset)
```

HD64180 New Opcodes (Z80 Extension)

New HD64180 Opcodes

```
ED 00 nn  IN0 B,(nn)    ED 01 nn  OUT0 (nn),B    ED 04     TST B
ED 08 nn  IN0 C,(nn)    ED 09 nn  OUT0 (nn),C    ED 0C     TST C
ED 10 nn  IN0 D,(nn)    ED 11 nn  OUT0 (nn),D    ED 14     TST D
ED 18 nn  IN0 E,(nn)    ED 19 nn  OUT0 (nn),E    ED 1C     TST E
ED 20 nn  IN0 H,(nn)    ED 21 nn  OUT0 (nn),H    ED 24     TST H
ED 28 nn  IN0 L,(nn)    ED 29 nn  OUT0 (nn),L    ED 2C     TST L
ED 30 nn  IN0 (nn)                               ED 34     TST (HL)
ED 38 nn  IN0 A,(nn)    ED 39 nn  OUT0 (nn),A    ED 3C     TST A
ED 4C     MULT BC       ED 83     OTIM           ED 64 nn  TST nn
ED 5C     MULT DE       ED 8B     OTDM           ED 70     IN (C)
```

```
  ED 7C     MULT SP       ED 9B     OTDMR          ED 76     SLP
```
On a real Z80, ED-4C/5C/6C/7C and ED-64/74 have been mirrors of NEG.
On a real Z80, ED-70 did the same (but was undocumented).
On a real Z80, ED-76 has been mirror of IM 2.
On a real Z80, ED-00..3F and ED-80..9F have acted as NOP.

Notes
IN0/OUT0/OTxMx same as IN/OUT/OTxx but with I/O-address bit8-15 forced 00h.
TST op: Test A,op. ;non-destructive AND (only flags changed)
TSTIO nn: Test Port[C],nn ;\hitachi lists BOTH definitions (page 75)
TSTIO nn: Test Port[nn],A ;/zilog also lists BOTH definitions (page 173,174)
TSTIO nn: Test Port[C],nn ;<-- this is reportedly the correct definition
MLT xy: xy=x*y ;unsigned multiply (flags=unchanged)
SLP (SLEEP) stops internal clock (including stopping DRAM refresh and DMAC).
IOSTOP: stops ASCI, CSI/O, PRT.

Z80 incompatible opcodes (according to Zilog's Z180 Application Note)
```
  Opcode    Z80                   Z180
  DAA       Checks Cy and A>99h   Checks Cy only? (when N=1)
  RLD/RRD   Sets flags for A      Sets flags for [HL]
```

Opcode Execution Time
Some opcodes are slightly faster as on real Z80. For example, some (not all) 4-cycle Z80 opcodes take only 3-cyles on HD64180.

Undefined Opcodes
On the HD64180, undefined opcodes are causing a TRAP exception (this feature cannot be disabled). So, while the real Z80 does have some useful (and some useless) undocumented opcodes, none (?) of these is working on HD64180 (except for the now-official ED-70 opcode).
The HD64180 datasheet doesn't list "SLL" as valid opcode.
The HD64180 datasheet doesn't list the "SET-and-LD" or "RES-and-LD" opcodes.
The HD64180 datasheet doesn't list opcodes with "IXL,IXH,IYL,IYH" operands, however, it does mention existence of "IXL" here and there (however, that seems to refer only to 16bit operations like "PUSH IX" (which do internally split 16bit IX into two 8bit units).
The HD64180 datasheet lists EX DE,HL with IX/IY-prefix as invalid.

NEWER INFO:
The HD64180 is actually trapping all undocumented opcodes, even those that are more or less commonly used on Z80 CPUs, ie. the HD64180 doesn't support accessing IX/IY 16bit registers as 8bit fragments (IXH,IXL,IYH,IYL), doesn't support "SLL" opcode, nor useless opcode mirrors (like alternate NEG/IM/RETN/RETI/NOP mirrors).

HD64180 Serial I/O Ports (ASCI and CSI/O)

Asynchronous Serial Communication Interfaces (ASCI) and Clocked Serial I/O (CSI/I)

XXX pg 51... 56

00h - CNTLA0 - ASCI Channel 0 Control Reg A (10h on Reset, bit3=var)
01h - CNTLA1 - ASCI Channel 1 Control Reg A (10h on Reset, bit3/bit4=var)
```
  7   MPE   RX Multi Processor Filter (0=RX all bytes, 1=RX flagged bytes)
  6   RE    RX Receiver Enable        (0=Disable, 1=Enable)
  5   TE    TX Transmitter Enable     (0=Disable, 1=Enable)
  4   /RTS0 for Ch0: Request to Send output (0=Low, 1=High) (/RTS pin)
      CKL1D for Ch1: CKA1 Clock Disable (CKA1/TEND pin)
  3   MPBR  Read:  RX Multi Processor Bit (Received Flag-Bit)
      EFR   Write: RX Error Flag Reset (0=Reset OVRN,PE,FE-Flags, 1=No Change)
  2   MOD2  Number of Data bits   (0=7bit, 1=8bit)
  1   MOD1  Number of Parity bits (0=None, 1=1bit) (only if MP=0)
  0   MOD0  Number of Stop bits   (0=1bit, 1=2bit)
```

02h - CNTLB0 - ASCI Channel 0 Control Reg B (07h on Reset, bit7/bit5=var)
03h - CNTLB1 - ASCI Channel 1 Control Reg B (07h on Reset, bit7=var)
```
  7   MPBT  TX Multi Processor Bit (Flag-Bit to be Transmitted)
  6   MP    Multiprocessor Mode (0=Off/Normal, 1=Add Flag-bit to all bytes)
  5   CTS   Read: /CTS-pin (0=Low, 1=High),
      PS    Write: Prescaler (0=Div10, 1=Div30)
  4   PEO   Parity Even/Odd (0=Even, 1=Odd) (ignored when MOD1=0 or MP=1)
  3   DR    Divide Ratio (0=Div16, 1=Div64)
  2-0 SS    Speed Select (0..6: "(PHI SHR N)", 7=External clock)
```
The baudrate is "SS div PS div DR" (or "External_Clock div DR").

04h - STAT0 - ASCI Channel 0 Status Register (00h on Reset, bit2/bit1=var)
05h - STAT1 - ASCI Channel 1 Status Register (02h on Reset)
```
  7   RDRF  RX Receive Data Register Full (0=No, 1=Yes)             (R)
  6   OVRN  RX Overrun Error (0=Okay, 1=Byte received while RDRF=1) (R)
  5   PE    RX Parity Error  (0=Okay, 1=Wrong Parity Bit)          (R)
  4   FE    RX Framing Error (0=Okay, 1=Wrop Stop Bit)             (R)
  3   RIE   RX Receive Interrupt Enable                (R/W)
  2   /DCD0 For Ch0: Data Carrier Detect (/DCD pin)    (R)
      CTS1E For Ch1: CTS input enable (/CTS pin)       (R/W)
  1   TDRE  TX Transmit Data Register Empty            (R)
  0   TIE   TX Transmit Interrupt Enable               (R/W)
```
Note: RDRD/TDRE can be used as DRQ signal for DMA channel 0.

06h - TDR0 - ASCI Channel 0 Transmit Data Register
07h - TDR1 - ASCI Channel 1 Transmit Data Register
08h - RDR0 - ASCI Channel 0 Receive Data Register
09h - RDR1 - ASCI Channel 1 Receive Data Register
```
  7-0  Data
```
The hardware can hold one byte in the data register (plus one byte currently processed in a separate shift register).

0Ah - CNTR - CSI/O Control Register (0Fh on Reset)

```
  7   EF    End Flag, completion of Receive/Transmit (0=No/Busy, 1=Yes/Ready)
  6   EIE   End Interrupt Enable (0=Disable, 1=Enable)
  5   RE    Receive Enable  (0=Off/Ready, 1=Start/Busy)
  4   TE    Transmit Enable (0=Off/Ready, 1=Start/Busy)
  3   -     Unused (should be all-ones)
  2-0 SS    Speed Select (0..6: "(20 shl N) clks per bit", 7=External clock)
```

The select "speed" is output on CKS pin (or input from CKS pin when selecting External clock). Bit7 is read-only (cleared when reading/writing TRDR).

0Bh - TRDR - CSI/O Transmit/Receive Data Register

```
  7-0  Data (8bit) (called TRDR by Hitachi, called TRD by Zilog)
```

Data is output on TXS pin, and input on RXS pin (both LSB first). Despite of the separate pins, one may NOT set RE and TE simultanoulsy (for whatever reason... or maybe it's meant to WORK ONLY if RX and TX are STARTED simultaneously). The RXS pin is also used as /CTS1 (for ASCI channel 1).

ASCI Multi Processor "Network" Feature

This feature allows to share the serial bus by multiple computers. Each byte is transferred with a "MPB" Multi Processor Flag Bit (located between Data and Stop bits) (Parity is forcefully disabled in Multi Processor Mode).
Assume broadcasting "Header+Data" Packets (with "Header" bytes flagged as MPB=1, and "Data" as MPB=0): The RX-Filter can select to receive only "Header" bytes, and, if the receiver treats itself to be addressed by the header, it can change the filter setting depending on whether it wants to receive/skip the following "Data" bytes.

HD64180 Timers (PRT and FRC)

Programmable Reload Timers (PRT) and Free Running Counter (FRC)

10h - TCR - Timer Control Register (00h on Reset)

```
  7   TIF1  Timer 1 Interrupt Flag (0=No, 1=Yes/Decrement reached 0000h) (R)
  6   TIF0  Timer 0 Interrupt Flag (0=No, 1=Yes/Decrement reached 0000h) (R)
  5   TIE1  Timer 1 Interrupt Enable (0=Disable, 1=Enable)
  4   TIE0  Timer 0 Interrupt Enable (0=Disable, 1=Enable)
  3-2 TOC   Timer 1 Output Control to A18-Pin (0=A18, 1=Toggled, 2=Low, 3=High)
  1   TDIE1 Timer 1 Decrement Enable (0=Stop, 1=Decrement; once every 20 clks)
  0   TDIE0 Timer 0 Decrement Enable (0=Stop, 1=Decrement; once every 20 clks)
```

TIF1 is reset when reading TCR or TMDR1L or TMDR1H.
TIF0 is reset when reading TCR or TMDR0L or TMDR0H.
The TOC bits control the A18/TOUT pin (it can be either A18 address line, or forced to Low or High, or "toggled": that is, inverted when TMDR1 decremts to 0.

0Ch - TMDR0L - Timer 0 Counter "Data" Register, Bit0-7 (FFh on Reset)
0Dh - TMDR0H - Timer 0 Counter "Data" Register, Bit8-15 (FFh on Reset)
0Eh - RLDR0L - Timer 0 Reload Register, Bit0-7 (FFh on Reset)
0Fh - RLDR0H - Timer 0 Reload Register, Bit8-15 (FFh on Reset)

Timer 0 counter/reload values. The counter is decremented once every 20 clks, and triggers IRQ and gets reloaded when reaching 0000h. Reading TMDR0L returns current timer LSB, and latches current timer MSB. Reading TMDR0H returns that LATCHED timer MSB. Accordingly reads should be always done in order LSB, MSB.
If the timer is stopped TMDR0L/TMDR0H can be written (and read) in any order.

14h - TMDR1L - Timer 1 Counter "Data" Register, Bit0-7 (FFh on Reset)
15h - TMDR1H - Timer 1 Counter "Data" Register, Bit8-15 (FFh on Reset)
16h - RLDR1L - Timer 1 Reload Register, Bit0-7 (FFh on Reset)
17h - RLDR1H - Timer 1 Reload Register, Bit8-15 (FFh on Reset)

Timer 1 counter/reload values. Same as for Timer 0 (see above).

18h - FRC - Free Running Counter (FFh on Reset)

```
  7-0 FRC   Free Running Counter (decremented every 10 clks)
```

This register should be read-only, writing to FRC may mess up DRAM refresh, ASCI and CSI/O baud rates.

HD64180 Direct Memory Access (DMA)

20h - SAR0L - DMA Channel 0 Source Address, Bit0-7 (Memory or I/O)
21h - SAR0H - DMA Channel 0 Source Address, Bit8-15 (Memory or I/O)
22h - SAR0B - DMA Channel 0 Source Address, Bit16-19 (Memory or DRQ)
23h - DAR0L - DMA Channel 0 Destination Address, Bit0-7 (Memory or I/O)
24h - DAR0H - DMA Channel 0 Destination Address, Bit8-15 (Memory or I/O)
25h - DAR0B - DMA Channel 0 Destination Address, Bit16-19 (Memory or DRQ)
26h - BCR0L - DMA Channel 0 Byte Count Register, Bit0-7
27h - BCR0H - DMA Channel 0 Byte Count Register, Bit8-15

DMA Channel 1 Source/Dest/Len. Direction can be Memory-to-Memory, Memory-to-I/O, I/O-to-Memory, or I/O-to-I/O, Memory-Address can be Fixed, Incrementing, or Decrementing, I/O-Address is Fixed (see DMODE Register).
For I/O transfers, Bit16-17 of SAR/DAR are selecting the DRQ type:

```
  00h DRQ by /DREQ0-Pin (normal case)
  01h DRQ by ASCI Channel 0 (RDRF-Bit for Source, or TDRE-Bit for Dest)
  02h DRQ by ASCI Channel 1 (RDRF-Bit for Source, or TDRE-Bit for Dest)
  03h Reserved
```

Memory-to-Memory DMA clock can be selected in MMOD bit ("Burst" pauses CPU until transfer is completed, "Cycle Steal" keeps the CPU running at roughly half-speed during DMA).

28h - MAR1L - DMA Channel 1 Memory Address, Bit0-7 (Source or Dest)
29h - MAR1H - DMA Channel 1 Memory Address, Bit8-15 (Source or Dest)
2Ah - MAR1B - DMA Channel 1 Memory Address, Bit16-19 (Source or Dest)
2Bh - IAR1L - DMA Channel 1 I/O Address, Bit0-7 (Dest or Source)
2Ch - IAR1H - DMA Channel 1 I/O Address, Bit8-15 (Dest or Source)
2Eh - BCR1L - DMA Channel 1 Byte Count Register, Bit0-7
2Fh - BCR1H - DMA Channel 1 Byte Count Register, Bit8-15

DMA Channel 1 Source/Dest/Len. Direction can be Memory-to-I/O or I/O-to-Memory, Memory-Address can be Incrementing or Decrementing, I/O-Address is Fixed (see DCNTL Register). DRQ is taken from /DREQ1-Pin.

30h - DSTAT - DMA "Status" Register (32h on Reset)
```
7   DE1   DMA Channel 1 Enable (0=Ready, 1=Start/Busy)
6   DE0   DMA Channel 0 Enable (0=Ready, 1=Start/Busy)
5   /DWE1 Writing to DE1 (0=Allowed, 1=Ignored, keep Bit7 unchanged)
4   /DWE0 Writing to DE0 (0=Allowed, 1=Ignored, keep Bit6 unchanged)
3   DIE1  DMA Channel 1 Interrupt Enable (0=Disable, 1=Enable)
2   DIE0  DMA Channel 0 Interrupt Enable (0=Disable, 1=Enable)
1   -     Unused (should be all-ones)
0   DME   DMA Main Enable
```

31h - DMODE - DMA Mode Register (E1h on Reset)
```
7-6 -     Unused (should be all-ones)
5-4 DM    DMA Channel 0 Dest (0=Mem/Inc, 1=Mem/Dec, 2=Mem/Fix, 3=IO/Fix)
3-2 SM    DMA Channel 0 Src  (0=Mem/Inc, 1=Mem/Dec, 2=Mem/Fix, 3=IO/Fix)
1   MMOD  DMA Channel 0 Mem-to-Mem Mode (0=Cycle Steal, 1=Burst)
0   -     Unused (should be all-ones)
```

32h - DCNTL - DMA/WAIT Control Register (F0h on Reset)
```
7-6 MW   Memory Waitstates (0..3 = 0..3)
5-4 IW   External I/O Waitstates (0..3 = 1..4) and /INT/LIR and more XXX
3   DMS1 DMA Channel 1 Sense /DREQ1-Pin (0=Sense Level, 1=Sense Edge)
2   DMS0 DMA Channel 0 Sense /DREQ0-Pin (0=Sense Level, 1=Sense Edge)
1   DIM1 DMA Channel 1 Src-to-Dest Direction (0=Mem-to-I/O, 1=I/O-to-Mem)
0   DIM0 DMA Channel 1 Memory-Step Direction (0=Increment, 1=Decrement)
```

Note
On some chip versions address bus is only 19bits, namely that does apply on 64pin chips (68pin/80pin chips should have 20bits). Regardless of the pin-outs, the extra bit might (maybe) exist internally on newer 64pin chips(?)

HD64180 Interrupts

Interrupts
```
Prio                                    Vector
0  /RES  Reset (non-maskable)           (PC=0000h, with TRAP=0 in ITC)
1  TRAP  Undefined Opcode (non-maskable) (PC=0000h, with TRAP=1 in ITC)
2  /NMI  Non-maskable Interrupt         (PC=0066h)
3  /INT0 Maskable Interrupt Level 0     (PC=[I*100h+databus], or PC=0038h)
4  /INT1 Maskable Interrupt Level 1     (PC=[I*100h+IL*20h+00h])
5  /INT2 Maskable Interrupt Level 2     (PC=[I*100h+IL*20h+02h])
6  Timer 0                              (PC=[I*100h+IL*20h+04h])
7  Timer 1                              (PC=[I*100h+IL*20h+06h])
8  DMA Channel 0 Ready                  (PC=[I*100h+IL*20h+08h])
9  DMA Channel 1 Ready                  (PC=[I*100h+IL*20h+0Ah])
10 Clocked Serial I/O Port (CSI/O)      (PC=[I*100h+IL*20h+0Ch])
11 Asynchronous SCI channel 0           (PC=[I*100h+IL*20h+0Eh])
12 Asynchronous SCI channel 1           (PC=[I*100h+IL*20h+10h])
Below whatever only (not HD64180 and not Z180)
?  Input Capture                        (PC=[I*100h+IL*20h+10h])
?  Output Compare                       (PC=[I*100h+IL*20h+12h])
?  Timer Overflow                       (PC=[I*100h+IL*20h+16h])
```
Note: "I" is a CPU-register (set via MOV I,A opcode). "IL" is new I/O port (set via OUT opcode). /INT0 works same as on real Z80 (and depends on mode set via IM 0/1/2 opcodes).

33h - IL - Interrupt Vector Low Register (00h on Reset)
```
7-5 IL   Bit7-5 of IM 2 Interrupt Vector Table Address
4-0 -    Unused (should be zero)
```

34h - ITC - INT/TRAP Control Register (39h on Reset)
```
7   TRAP Undefined Opcode occurred (0=No, 1=Yes)
6   UFO  Addr of Undef Opcode (aka Undefined Fetch Object) (0=PC-1, 1=PC-2)
5-3 -    Unused (should be all-ones)
2   ITE2 Interrupt /INT2 Enable (0=Disable, 1=Enable)
1   ITE1 Interrupt /INT1 Enable (0=Disable, 1=Enable)
0   ITE0 Interrupt /INT0 Enable (0=Disable, 1=Enable)
```
TRAP gets set upon Undefined Opcodes (TRAP and RESET are both using vector 0000h, the TRAP bit allows to sense if the vector was called by Reset or Undef Opcode). The TRAP bit can be cleared by software by writing "0" to it (however, software cannot write "1" to it).

HD64180 Memory Mapping and Control

Memory Managment Unit (MMU)
The Memory Managment Unit translates "virtual" 16bit CPU memory addresses to "physical" 19bit address bus.
```
0000h -------> +--------------------+
               | Common Area 0      | Phys19bit = Virt16bit + 00000h
BA*1000h ----> +--------------------+
               | Bank Area          | Phys19bit = Virt16bit + BBR*1000h
CA*1000h ----> +--------------------+
               | Common Area 1      | Phys19bit = Virt16bit + CBR*1000h
FFFFh -------> +--------------------+
```
The 16bit CPU address space is divided into three areas (of which, the first two areas can be 0 bytes in size: BA=0 disables Common Area 0, CA=CB disables Bank Area).

38h - CBR - MMU Common Base Register (Common Area 1) (00h on Reset)
39h - BBR - MMU Bank Base Register (Bank Area) (00h on Reset)
```
7   Unused (should be zero) (but, used on chips with 20bit address bus)
0-6 Base in 4K-units within "physical" 19bit 512K address space
```

3Ah - CBAR - MMU Common/Bank Area Register (F0h on Reset)
```
4-7 CA Start of Common Area 1 (End of Bank Area) (0Fh upon Reset)
0-3 BA Start of Bank Area (End of Common Area 0) (00h upon Reset)
```
This is in 4K-units within the "virtual" 16bit 64K address space. Results on CA<CB are undefined.

```
  7   REFE  DRAM Refresh Enable (0=Disable, 1=Enable)
  6   REFW  DRAM Refresh Wait (0=Two Clocks, 1=Three Clks)
  5-2 -     Unused (should be all-ones)
  1-0 CYC   DRAM Refresh Interval (0..3=10,20,40,80 states)
```

Note: The hardware outputs an 8bit Refresh address on A0..A7. A classic Z80 did output only 7bits, via using the CPU's "R" register (accessible with MOV A,R and MOV R,A opcodes). The HD64180 does still increment lower 7bit of "R" in same/similar fashion as on Z80, but, as far as I understand, without affecting any bits of the actual refresh address (and vice-versa, without the RCR-register settings affecting the way how "R" gets incremented).

3Fh - ICR - I/O Control Register (1Fh on Reset)

```
  7-6 IOA   Base Address of Internal I/O ports (0..3=0000h,0040h,0080h,00C0h)
  5   IOSTP Stop Internal ASCI, CSI/O, PRT-Timers (0=No, 1=Pause)
  4-0 -     Unused (all ones on Reset)
```

Note: There is a "z180.h" file in the internet that claims that ICR "does not move" (ie. that the "IOA" bits affect only Port 00h-3Eh, but not Port 3Fh itself). Unknow where that info comes from, and unknown if it's correct (the HD64180 and Z180 datasheets do not mention that effect).

Memory Address Bus Width

According to official specs, the address bus is 19bits wide (although the same specs claim that it can address up to 1Mbyte, which would require 20bits, unknown how that could work). [The 68pin and 80pin chip versions do actually have a new A19 pin, which doesn't exist on 64pin chips]
Observe that A18 can be misused as Square-Wave output or as General-purpose output (see Timer chapter); when using that feature, one should normally connect only A0..A17 to memory address bus - otherwise, if A18 is wired to memory, the feature would cause the physical address to be ANDed with 3FFFFh or ORed with 40000h (this "allows" to some futher, but rather useless, bankswitching).

Waitstate Control

For Memory and I/O Waitstate control, see DCNTL register (in DMA chapter).

DMA

DMA transfers can directly access 19bit addresses, without using the MMU.

Unused Bits

Several internal registers contain unused bits. According to the datasheet, upon reset, these bits are set to all-ones, or all-zero (the setting varies from register to register). Unknown if it's possible and/or allowed to change these bits.

Reserved Registers

Registers 11h-13h, 19h-1Fh, 2Dh, 35h, 37h, 3Bh-3Eh are Reserved. Unknown if it's possible and/or allowed to read/write these registers.

HD64180 Extensions

Port 3Eh on Z8x180 only (not HD64180).
Port 12h..13h,1Ah..1Fh,2Dh on Z8S180/Z8L180 only (not Z80180/HD64180).

3Eh - OMCR - Operation Mode Control - Z180 only (not HD64180) (FFh on Reset)

```
  7   M1E   /M1 Enable           (0=Z180, 1=HD64180; Problems with RETI)  (R/W)
  6   /M1TE /M1 Temporary Enable (0=Z180, 1=HD64180; Problems with Z80PIO)  (W)
  5   /IOC  I/O Compatibility    (0=Z180, 1=HD64180; Delayed falling /WR) (R/W)
  4-0 -     Unused (should be all-ones)
```

Allows to fix some signal & timing glitches of the HD64180 (or to maintain them for compatibility with HD64180 based designs).

12h - ASEXT0 - ASCI Channel 0 Extension Control Reg 0 (00h on Reset)
13h - ASEXT1 - ASCI Channel 1 Extension Control Reg 1 (00h on Reset)

```
  7   RDRF Interrupt Inhibit
  6   DCD0 Disable    ;\ASCI Channel 0 only (not Channel 1)
  5   CTS0 Disable    ;/
  4   X1 Bit Clk ASCI
  3   BRG Mode (Time Constant based Baud Rate Generator)
  2   Break Feature Enable
  1   Break Detect (RO)
  0   Send Break
```

1Ah - ASTC0L - ASCI Channel 0 Time Constant, Bit0-7 (00h on Reset)
1Bh - ASTC0H - ASCI Channel 0 Time Constant, Bit8-15 (00h on Reset)
1Ch - ASTC1L - ASCI Channel 1 Time Constant, Bit0-7 (00h on Reset)
1Dh - ASTC1H - ASCI Channel 1 Time Constant, Bit8-15 (00h on Reset)

16bit Time Constants (see BRG bit in ASEXT0/ASEXT1).

1Eh - CMR - Clock Multiplier Register (7Fh or so on Reset)

```
  7   X2  Enable X2 Clock Multiplier Mode (0=Disable, 1=Enable)
  6-0 -   Unused (should be all-ones) (or so)
```

Purpose undocumented, maybe doubles the CPU speed (and/or Timer or ASCI or whatever speeds).

1Fh - CCR - CPU Control Register (00h on Reset)

```
  7  Clock Divide (0=XTAL/2, 1=XTAL/1)
  6  Standby/Idle Mode, Bit1
  5  BREXT     (0=Ignore BUSREQ in Standby/Idle, Exit Standby/Idle on BUSREQ)
  4  LNPHI     (0=Standard Drive, 1=33% Drive on EXTPHI Clock)
  3  Standby/Idle Mode, Bit0
  2  LNIO      (0=Standard Drive, 1=33% Drive on certain external I/O)
  1  LNCPUCTL  (0=Standard Drive, 1=33% Drive on CPU control signals)
  0  LNAD/DATA (0=Standard Drive, 1=33% Drive on A10-A0, D7-D0)
```

Standby/Idle Mode is combined of CCR bit6/bit3 (0=No Standby, 1=Idle after Sleep, 2=Standby after Sleep, 3=Standby after Sleep with 64 Cycle Exit Quick Recovery).

2Dh - IAR1B - DMA "I/O Address Ch 1" (Absurde name) (00h on Reset)

```
  7   Alternating Channels
  6   Currently selected DMA channel when Bit7=1
  5-4 Unused (should be zero) (must be 0)
  3   TOUT/DREQ-Pin (0=DREQ Input, 1=TOUT Output)
  2-0 DMA Channel 1 (0=TOUT/DREQ, 1=ASCI0, 2=ASCI1, 3=ESCC, 7=PIA)
```

SNES Decompression Formats

Nintendo-specific Compression Overall Format (BSX and SFC-Box)
Compressed data consists of "code/length" pairs encoded in 1 or 2 bytes:
```
cccNnnnn           --> code (ccc=0..6) len 5bit (Nnnnn+1)
111cccNn.nnnnnnnn  --> code (ccc=0..7) len 10bit (Nnnnnnnnnn+1)
11111111           --> end code (FFh)
```
The "code/length" pairs are then follwed by "src" data, or "disp" offsets (depending on the "ccc" codes). The meaning of the "ccc" codes varies from program to program (see below for how they are used by BSX and SFC-Box).
Note: As seen above, ccc=7 works only with 10bit len (not 5bit len) and only with max len=2FFh+1 (not 3FFh+1).

Nintendo-specific Compression Codes (BSX) (Satellaview)
Used to decompress various data (including custom person OBJs in the Directory packet). The decompression functions are at 80939Fh (to RAM) and 80951Eh (to VRAM) in the BSX-BIOS. The meaning of the "ccc" codes is:
```
0  Copy_bytes_from_src
1  Fill_byte_from_src
2  Fill_word_from_src
3  Fill_incrementing_byte_from_src
4  Copy_bytes_from_dest_base_plus_16bit_disp
5  Copy_bytes_from_dest_base_plus_16bit_disp_with_invert
6  Copy_bytes_from_current_dest_addr_minus_8bit_disp
7  Copy_bytes_from_current_dest_addr_minus_8bit_disp_with_invert
```
For all codes (including ccc=2), len is the number of BYTEs to be copied/filled. For ccc=4..6, the code is followed by a 16bit offset in LITTLE-ENDIAN format. For ccc=5/7, copied data is inverted (XORed with FFh).

Nintendo-specific Compression Codes (SFC-Box) (Super Famicom Box)
Used (among others) to decompress the Title-Bitmaps in "GROM" EPROMs, the decompression function is at 0088A2h in the "ATROM" menu program. The meaning of the "ccc" codes is:
```
0  Copy_bytes_from_src
1  Fill_byte_from_src
2  Fill_word_from_src
3  Fill_incrementing_byte_from_src
4  Copy_bytes_from_dest_base_plus_16bit_disp
5  Copy_bytes_from_dest_base_plus_16bit_disp_with_xflip
6  Copy_bytes_from_dest_base_plus_16bit_disp_with_yflip
7  Unused (same as ccc=4)
```
For ccc=2, len is the number of WORDs to be filled, for all other codes, it's the number of BYTEs to be copied/filled. For ccc=4..7, the code is followed by a 16bit offset in BIG-ENDIAN format. For ccc=5 (xflip), bit-order of all bytes is reversed (bit0/1/2/3 <--> bit7/6/5/4). For ccc=6 (yflip), reading starts at dest_base+disp (as usually), but the read-address is then decremented after each byte-transfer (instead of incremented).

SNES Decompression Hardware
The APU automatically decompresses BRR-encoded audio samples (4bit to 15bit ADPCM, roughly similar to CD-XA format). Cartridges with SPC7110 or S-DD1 chips can decompress (roughly JPEG-style) video data, and convert it to SNES bit-plane format. Cartridges with SA-1 chips include a "Variable-Length Bit Processing" feature for reading "N" bits from a compressed bit-stream.

SNES Unpredictable Things

Open Bus
Garbage appears on reads from unused addresses (see below), on reads from registers that contain less than 8 bits (see more below), and on reads from (most) write-only registers (see even more below). The received data is the value being previously output on the data bus. In most cases, that is the last byte of the opcode (direct reads), or the upper byte of indirect address (indirect reads), ie.
```
LDA IIJJ,Y   aka MOV A,[IIJJ+Y]  --> garbage = II
LDA (NN),Y   aka MOV A,[[NN]+Y]  --> garbage = [NN+1]
```
When using General Purpose DMA, it'd be probably rely on the store opcode that has started the DMA (in similar fashion as above loads). In case of HDMA things would be most unpredictable, garbage would be whatever current-opcode related value. And, if two DMA transfers follow directly after each other, garbage would probably come from the previous DMA.
Also, in 6502-mode (and maybe also in 65816-mode), the CPU may insert dummy-fetches from unintended addresses on page-wraps?

Unused Addresses in the System Region
```
Address       Size
2000h..20FFh  100h    ;unused addresses
2181h..2183h  3       ;write-only WRAM Address registers
2184h..21FFh  7Ch     ;unused addresses in the B-BUS region
2200h..3FFFh  1E00h   ;unused addresses
4000h..4015h  16h     ;unused slow-CPU Ports
4018h..41FFh  1E8h    ;unused slow-CPU Ports
4200h..420Dh  0Eh     ;write-only CPU Ports
420Eh..420Fh  2       ;unused CPU Ports
4220h..42FFh  E0h     ;unused CPU Ports
43xCh..43xEh  3*8     ;unused DMA Ports
4380h..7FFFh  3C80h   ;unused/expansion area
```
So, of the total of 8000h bytes, a large number of 5EF4h is left unused.
Ports 2144h..217Fh are APU mirrors, NOT open bus.

Unused bits (in Ports with less than 8 used bits)
```
Addr  Mask Name    Unused Bits
4016h FCh  JOYA    Bit7-2 are open bus
4017h E0h  JOYB    Bit7-5 are open bus
4210h 70h  RDNMI   Bit6-4 are open bus
4211h 7Fh  TIMEUP  Bit6-0 are open bus
4212h 3Eh  HVBJOY  Bit5-1 are open bus
```

PPU1 Open Bus
PPU1 Open Bus is relies on the value most recently read from Ports 2134h-2136h, 2138h-213Ah, 213Eh. This memorized value shows up on later reads from read-only Ports 21x4h..21x6h and 21x8h..21xAh (with x=0,1,2) (in all 8bits), as well as in Port 213Eh.Bit4.

PPU2 Open Bus
PPU2 Open Bus is relies on the value most recently read from Ports 213Bh-213Dh, 213Fh. This memorized value shows up on later reads from Port 213Bh.2nd_read.Bit7, 213Ch/213Dh.2nd_read.Bits7-1, and Port 213Fh.Bit5.

PPU Normal Open Bus
Other write-only PPU registers & the HV-latch "strobe" register are acting like "normal" CPU Open Bus (ie. usually returning the most recent opcode byte). These are 21x0h..21x3h, 21x7h (with x=0,1,2,3), and 21xBh..21xFh (with x=0,1,2).

Open Bus for DMA
DMA cannot read from most I/O ports, giving it some additional open bus areas:
```
  2100h-21FFh  Open Bus (when used as A-Bus) (of course they work as B-Bus)
  4000h-41FFh  Open Bus (name 4016h/4017h cannot be read)
         actually, 4017h <does> return bit4-2 set (1=GNDed joypad input)
  4210h-421Fh  These do work (the only I/O ports that are not open bus)
  4300h-437Fh  Special Open Bus (DMA registers, may return [PC] instead of FFh)
```
For DMA reads, one may expect the same garbage as for CPU reads (ie. when starting a DMA via "MOV [420Bh],A", one would expect 42h as open bus value). However, it takes a few cycles before the DMA transfer does actually start, during that time the hardware "forgets" the 42h value, and instread, DMA does always read FFh (High-Z) as open bus value. With two exceptions: If DMA wraps from an used to unused address (eg. from 1FFFh/WRAM to 2000h/unused) then the first "unused" byte will be same as the last "used" byte, thereafter, it forgets that value (and returns FFh on further unused addresses).
The other exception is if DMA <starts> at 4300h..437Fh: In this case it will read [PC] for that region (and will also "memorize" it when reaching the first unused address at 4380h, and then returns FFh for 4381h and up). For example: "MOV [420Bh],A" followed by "ADC A,33h" would return 69h (the first byte of the ADC opcode). This effect occurs only if the transfer <starts> in that region, ie. if starts below that area, and does then wrap from 42FFh to 4300h, then it returns FFh instead of 69h. (The reason of this special effect is probably that the DMA somehow "ignores the databus", so external HIGH-Z levels (like from XBOO cable or Cartridge) cannot drag the "memorized" value to HIGH.)
EDIT: The above "memorize for next ONE unused address" applies only when XBOO cable is connected (which pulls the databus to HIGH rather quickly). Without XBOO cable (and without cartridge connected) the "memorized" value may last for the next 2000h (!) unused addresses, then it may slowly get corrupted (some bits going to HIGH state, until, after some more time, all bits are HIGH).
Actually, it seems to last even longer than 2000h -- possibly forever (until DMA ends, or until it reaches a used address).

SPC700 Division Overflow/Result (DIV YA,X opcode)
The overall division mechanism (with and without overflows) is:
```
  H = (X AND 0Fh)<=(Y AND 0Fh)   ;half carry flag (odd dirt effect)
  Temp = YA
  FOR i=1 TO 9
    Temp=Temp*2                              ;\rotate within 17bits
    IF Temp AND 20000h THEN Temp=(Temp XOR 20001h)  ;/
    IF Temp>=(X*200h)  THEN Temp=(Temp XOR 1)
    IF Temp AND 1      THEN Temp=(Temp-(X*200h)) AND 1FFFFh
  NEXT i
  A = (Temp AND FFh) ;result.bit7-0
  V = (Temp.Bit8=1)  ;result.bit8
  Y = (Temp/200h)    ;remainder (aka temp.bit9-16)
  N = (A.Bit7=1)     ;sign-flag (on result.bit7) (though division is unsigned)
  Z = (A=00h)        ;zero-flag (on result.bit7-0)
```
That is, normally (when result = 0000h..00FFh):
```
  A=YA/X, Y=YA MOD X, N=ResultingA.Bit7, Z=(ResultingA=00h), V=0, H=(see above)
```
An intact 9bit result can be read from V:A (when result = 0000h..01FFh). Otherwise return values are useless garbage (when result = 0200h..Infinite).

SNES Timings

SNES Timing Oscillators

NTSC Timings
```
  NTSC crystal      21.4772700MHz (X1, type number D214K1)
  NTSC color clock  3.57954500MHz (21.47727MHz/6)  (generated by PPU2 chip)
  NTSC master clock 21.4772700MHz (21.47727MHz/1)  (without multiplier/divider)
  NTSC dot clock    5.36931750MHz (21.47727MHz/4)  (generated by PPU chip)
  NTSC cpu clock    3.57954500MHz (21.47727MHz/6)  (without waitstates)
  NTSC cpu clock    2.68465875MHz (21.47727MHz/8)  (short waitstates)
  NTSC cpu clock    1.78977250MHz (21.47727MHz/12) (joypad waitstates)
  NTSC frame rate   60.09880627Hz (21.477270MHz/(262*1364-4/2))
  NTSC interlace    30.xxxxxxxxHz (21.477270MHz/(525*1364))
```

PAL Timings
```
  PAL crystal       17.7344750MHz (X1, type number D177F2)
  PAL color clock   4.43361875MHz (17.7344750MHz/4)   (generated by S-CLK chip)
  PAL master clock  21.2813700MHz (17.7344750MHz*6/5) (generated by S-CLK chip)
  PAL dot clock     5.32034250MHz (21.2813700MHz/4)   (generated by PPU chip)
  PAL cpu clock     3.54689500MHz (21.2813700MHz/6)   (without waitstates)
  PAL cpu clock     2.66017125MHz (21.2813700MHz/8)   (short waitstates)
  PAL cpu clock     1.77344750MHz (21.2813700MHz/12)  (joypad waitstates)
  PAL frame rate    50.00697891Hz (21.281370MHz/(312*1364))
  PAL interlace     25.xxxxxxxxHz (21.281370MHz/(625*1364+4/2))
```

APU Timings
```
  APU oscillator    24.576MHz (X2, type number 24.57MX)
  DSP sample rate   32000Hz   (24.576MHz/24/32)
  SPC700 cpu clock  1.024MHz  (24.576MHz/24)
  SPC700 timer 0+1  8000Hz    (24.576MHz/24/128)
  SPC700 timer 2    64000Hz   (24.576MHz/24/16)
  CIC clock         3.072MHz  (24.576MHz/8)
  Expansion Port    8.192MHz  (24.576MHz/3)
```

CPU Clock Notes
CPU Clock cycles (opcode fetches, data transfers, and internal cycles) are usually clocked at 3.5MHz or 2.6MHz (or a mixup thereof).
```
  3.5MHz   Used for Fast ROM, most I/O ports, and internal CPU cycles
  2.6MHz   Used for Slow ROM, for WRAM, and for DMA/HDMA transfers
  1.7MHz   Used only for (some) Joypad I/O Ports
```
The CPU is paused for 40 master cycles (per 1364 cycle scanline) for memory REFRESH purposes, effectively making the CPU around 3% slower. The CPU is

also paused when using DMA/HDMA transfers.
Nintendo specifies the following ROM timings to be required:

```
3.5MHz   use 120ns or faster ROM/EPROMs
2.6MHz   use 200ns or faster ROM/EPROMs
```

Dot Clock Notes
The above values apply for the drawing period, in the hblank period some cycles are a bit longer. This "stuttering" effect appears also on the dotclk output on expansion port.

External Oscillators (in Cartridges)

```
SGB       <master>
SGB2      20.9MHz          External oscillator (located on PCBs solder-side)
SPC7110   <master>
CX4       20.000MHz
ST010     22.000MHz (or reportedly effective 20MHz/2 ?)
ST011     15.000MHz
ST018     21.44MHz
DSPn      ? MHz
MC1       <master>
GSU1      21.4 MHz
GSU2      21.44MHz
SA-1      <master>
BS-X      18.432MHz        Satellaview Receiver Unit (on expansion port)
RTC-4513  32.768kHz        On-chip 32.768kHz quartz crystal in RTC chip
S-3520    32.768kHz        External 32.768kHz quartz crystal (SFC-Box)
S-RTC     ? kHz            External unknown-frequency crystal
```

Master = 21.4772700MHz (NTSC), or 21.2813700MHz (PAL).

SNES Timing H/V Counters

Horizontal Timings

```
Scanline Length        1364 master cycles (341 dot cycles)
  Except, Line F0h in Field.Bit=1 of Interlace: 1360 master cycles
Refresh (per scanline)   40 master cycles (10 dot cycles)

50*312*1364 = 21.278400 MHz   // 21.281370MHz/(312*1364) = 50.00697891 Hz
60*262*1364 = 21.442080 MHz   // 21.477270MHz/(262*1364-2) = 60.09880627 Hz
```

Long and Short Scanlines
A normal scanline is 1364 master cycles long. But, there are two special cases, in which lines are 4 cycles longer or shorter:

```
Short Line --> at 60Hz frame rate + interlace=off + field=1 + line=240
Long Line  --> at 50Hz frame rate + interlace=on + field=1 + line=311
(in both cases, the selected picture size, 224 or 239 lines, doesn't matter)
```

Technically, the effects work as so:

```
Normal Line : 1364 cycles, 340 dots (0-339), four dots are 5-cycles long
Long Line   : 1368 cycles, 341 dots (0-340), four dots are 5-cycles long
Short Line  : 1360 cycles, 340 dots (0-339), all dots are 4-cycles long
```

Glitch: The long scanline is placed in the last line (directly after the Hsync for line 0, thus shifting the Hsync position of Line 1, ie. of the first line of the drawing period), accordingly, the upper some scanlines in interlaced 50Hz mode are visibly shifted to the right (by around one pixel), until after a handful of scanlines the picture stablizes on the new hsync position (ie. trying to display a vertical line will appear a little curved).

Long and Short Scanlines (Purpose)
The Scanline Rate doesn't match up with the PAL/NTSC Color Clocks, so, for example, a red rectangle on black background will look like so:

```
RGB-Output             Composite-Output        Composite-Output
Flawless               Static-Error            Flimmering-Error
RRRRRRRRRRRRRRRR       RRRRRRRRRRRRRRRR        rRRRRRRRRRRRRRRRr
RRRRRRRRRRRRRRRR        RRRRRRRRRRRRRRRR       rrRRRRRRRRRRRRRRrr
RRRRRRRRRRRRRRRR         RRRRRRRRRRRRRRRR       rRRRRRRRRRRRRRRRr
RRRRRRRRRRRRRRRR       RRRRRRRRRRRRRRRR        rRRRRRRRRRRRRRRRr
RRRRRRRRRRRRRRRR        RRRRRRRRRRRRRRRR       rrRRRRRRRRRRRRRRrr
RRRRRRRRRRRRRRRR         RRRRRRRRRRRRRRRR       rRRRRRRRRRRRRRRRr
```

Inserting the long/short scanlines does synchronize the Frame Rate with the PAL/NTSC color clocks:

```
PAL Mode        Master Clocks (21MHz)       Color Clocks (PAL:4.4MHz)
50Hz Normal     425568 (312*1364)           88660 (425568/6*5/4)
50Hz Interlace  426936 (313*1364+4)         88945 (426936/6*5/4)
NTSC Mode       Master Clocks (21MHz)       Color Clocks (NTSC:3.5MHz)
30Hz Normal     714732 ((262+262)*1364-4)   119122 (714732/6)
30Hz Interlace  716100 ((262+263)*1364)     119350 (716100/6)
```

The result is that the composite video output is producing the "Static Error" effect (in the above example, the rectangle has sawtooth-edges). And the "Flimmering" effect is avoided (which would have blurry edges, and which would also appear as if the edges were wandering up) (Note: The flimmering effect can be seen when switching a modded 50Hz PAL console to 60Hz mode).

SNES Timing H/V Events

Summary of Vertical Timings

```
V=0              End of Vblank, toggle Field, prefetch OBJs of 1st scanline
V=0..224/239     Perform HDMA transfers (before each line & after last line)
V=1              Begin of Drawing Period
V=225/240        Begin of Vblank Period (NMI, joypad read, reload OAMADD)
V=240            Short scanline in Non-interlaced 60Hz field=1
V=311            Long scanline in Interlaced 50Hz field=1
V=261/311        Last Line (in normal frames)
V=262/312        Extra scanline (occurs only in Interlace Field=0)
V=VTIME          Trigger V-IRQ or HV-IRQ
```

Detailed List (H=Horizontal, V=Vertical, F=Field)

```
H=0, V=0, F=0         SNES starts at this time after /RESET
H=0, V=0              clear Vblank flag, and reset NMI flag (auto ack)
H=0, V=225            set Vblank flag
H=0.5, V=225          set NMI flag
H=1                   clear hblank flag
H=1, V=0              toggle interlace FIELD flag
```

```
H=HTIME+3.5           H-IRQ
H=2.5, V=VTIME        V-IRQ   (or HV-IRQ with HTIME=0)
H=HTIME+3.5, V=VTIME  HV-IRQ  (when HTIME=1..339)
H=6, V=0              reload HDMA registers
H=10, V=225           reload OAMADD
H=22-277(?), V=1-224  draw picture
H=32.5..95.5, V=225   around here, joypad read begins (duration 4224 clks)
H=133.5               around here, REFRESH begins (duration 40 clks/10 dots)
H=274                 set hblank flag
H=278, V=0..224       perform HDMA transfers
H=323,327             seen as long-PPU-dots (but not as long-CPU-dots)
H=323,327, V=240, F=1 seen as normal-PPU-dots (in short scanline 240) (60Hz)
H=339                 this is last PPU-dot (in normal and short scanlines)
H=340, V=311, F=1     this is last PPU-dot in long scanlines (50Hz+Interlace)
CPU.H=339             this is last CPU-dot (in normal scanlines)
CPU.H=338, V=240, F=1 this is last CPU-dot (in short scanlines)
CPU.H=340, V=311, F=1 this is last CPU-dot (in long scanlines)
H=0?, V=0             reset OBJ overflow flags in 213Eh (only if not f-blank)
H=0?+INDEX*2, V=YLOC  set OBJ overflow bit6 (too many OBJs in next line)
H=0?, V=YLOC+1        set OBJ overflow bit7 (too many OBJ pix in this line)
...?
```

xxx joypad read
xxx reload mosaic h/v counter (at some point during vblank)
xxx count mosaic v counter (at some point in each scanline)

Note that a PAL TV-set can display around 264 lines (about 25 more than supported by the 239-line mode).

Note that a superscope pointing at pixel (X, Y) on the screen will latch
approximately dot X+40 on scanline Y+1.

PPU H-Counter-Latch Quantities
When latching PPU H/V-latches via reading [2137h] by software, 341*4 times at evenly spread locations, one will statistically get following H values:

```
0..132    4 times (normal)
133       3 times (occurs sometimes at H=133.5, and always at H=133.0)
134       1 time  (occurs sometimes at H=134.x)
135..142  never   (refresh is busy, cpu is stopped)
143       1 time  (occurs sometimes at H=143.x)
144       3 times (occurs sometimes at H=144.0, and always at H=144.5)
145..322  4 times (normal)
323       6 times (seen as long dot) (or 5.99 times if NTSC+InterlaceOff)
324..326  4 times (normal)
327       6 times (seen as long dot) (or 5.99 times if NTSC+InterlaceOff)
328..339  4 times (normal)
340       never   (doesn't exist) (or 0.01 times if PAL+InterlaceOn)
341-511   never   (doesn't exist)
```

For the 1 and 3 times effect, one would expect the 40 clk refresh as so:

```
--- H=133.5-->x<--H=134.0 ----------------- H=143.5-->x<--H=144.0 ---
ccccccccccccccRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRccccccccccccccc
```

but, sometimes (randomly at 50:50 chance) it occurs somewhat like so:

```
ccccccccccccRRccRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRccRRcccccccccccc
```

whereas "c"=CPU, "R"=Refresh (ie. sometimes, the CPU wakes up within the Refresh period). Unknown why & when exactly that stuttering refresh occurs.
Opcodes may begin before Refresh, and end after Refresh (rather than forcefully finishing the Opcode before starting Refresh), so above statistics are same for latching via "MOV A,[2137h]" (4 cycles) and "MOV A,[002137h]" (5 cycles) opcodes.
Writing [4201h] (instead of reading [3137h]) shifts the visible refresh-related H values (H=136..143=never, H=135/144 once, H=134/H=145 thrice), but obviously doesn't alter the amounts of visible PPU-related H values.

SNES Timing PPU Memory Accesses

Below is some info/guesses on what is happening during the 1364 master cycles of a scanline. Plus some info/guesses on if/when/why it is (or isn't) possible the change VRAM/CGRAM/OAM outside of V-Blank or Forced-Blank.

PPU VRAM Load
VRAM access time is 4 master cycles per word (aka 1 dot-cycle per word). In each line, the PPU fetches 34*2 words (for 34 OBJ tiles of 8pix/4bpp), plus 33*8 words (for BgMap+BgTiles) (eg. 33*4 BgMap entries plus 33*4 BgTiles at 2bpp in BG Mode 0) (or less BG layers, but more bpp per tile in other BG Modes). As a whole, 4 master cycles per 34*2 + 33*8 words sums up to 1328 master cycles (or possibly a bit more if accesses during "long dots" should happen to take 5 master cycles per word). 1328 would leave 36 unused master cycles per scanline (or maybe MORE unused cycles if some BG layers are disabled? and/or if less than 34 OBJ tiles are located in the scanline? unknown if VRAM can be accessed during any such unused cycles? as far as known, existing games aren't trying to do so).

PPU Palette Load
Not much known about CGRAM access timings. There would be some theories:

```
1) Maybe 512 accesses/line (for frontmost main/sub-screen pixels)
2) Maybe 1296 accesses/line (for 256*4 BG pixels + 34*8 OBJ pixels)
3) Maybe 1360 accesses/line (for 33*4*8 BG pixels + 34*8 OBJ pixels)
```

For the 1296/1360 accesses therories, CGRAM would be inaccessible for most of the 1364 cycles.
In practice, it is possible to change CGRAM during certain timespots in the Hblank-period (not during the <whole> Hblank period, but it works <sometimes>, though doing so may require some research - and may end up with more or less fragile timings). As for when/why it is working: Maybe there some totally unused cycles, or it depends on how many OBJs are displayed in the current scanline, possibly also on the number of used and/or enabled BG layers (and thereby, maybe also allowing to change CGRAM outside of Hblank during BG-drawing period?).

PPU OAM load
OAM handling is done in three steps:

```
1) scan 128 entries (collect max 32 entries per line)
2) scan 32 entries (collect max 34 tiles per line)
3) scan 34 entries (collect max 34x8 pixels per line)
```

OAM-Access time for Step 1 is 128*2 dots aka 128*8 master cycles (as seen in STAT77.Bit6). OAM-Access times for Steps 2 and 3 is unknown (might be some cycles per entry... or might be NULL cycles; in case OAM entries were copied to separate buffer during previous Step... and/or NULL cycles if no OBJs are in current scanline).
So, aside from the 256 known/used dot-cycles, there may (or may not) be up to 84 unused dot-cycles... possibly allowing to change OAM during Hblank(?).
Note: Mario Kart is using Forced Blank to change OAM in middle of screen. Observe that the OAM address in Port 2102h is scattered during drawing period.

SNES Pinouts

External Connector Pinouts
SNES Controllers Pinouts
SNES Audio/Video Connector Pinouts
SNES Power Supply
SNES Expansion Port (EXT) Pinouts
SNES Cartridge Slot Pinouts

Chipset Pinouts
SNES Chipset
SNES Pinouts CPU Chip
SNES Pinouts PPU Chips
SNES Pinouts APU Chips
SNES Pinouts ROM Chips
SNES Pinouts RAM Chips
SNES Pinouts CIC Chips
SNES Pinouts MAD Chips
SNES Pinouts Misc Chips
SNES Pinouts GSU Chips
SNES Pinouts CX4 Chip
SNES Pinouts SA1 Chip
SNES Pinouts Decompression Chips
SNES Pinouts BSX Connectors

Mods
SNES Common Mods
SNES Controller Mods
SNES Xboo Upload (WRAM Boot)

SNES Controllers Pinouts

Joypads (2)

```
Pin  Dir  Port 1         Port2              _____________ _________________
1    -    VCC +5VDC      VCC +5VDC        / 7   6   5  | 4   3   2   1  |
2    Out  JOY-1/3 Clock  JOY-2/4 Clock    | GND IO6 IN3 | IN1 STB CK1 VCC | 1
3    Out  JOY-STROBE     JOY-STROBE        \_____________|_________________|
4    In   JOY-1 Data     JOY-2 Data         _____________ _________________
5    In   JOY-3 Data     JOY-4 Data       / 7  PEN  5  | 4   3   2   1  |
6    I/O  I/O bit6       I/O bit7, Pen    | GND IO7 IN4 | IN2 STB CK2 VCC | 2
7    -    GND            GND               \_____________|_________________|
```

Pin 6 on Port 2 is shared for I/O and Lightpen input.

Internal Connector
The two joypad connectors (and power LED) are located on a small daughterboard, which connects to the mainboard via an 11pin connector:

```
1 VCC
2 IO6       ;-pad1
3 IO7 / pen ;\
4 IN2       ; pad2
5 IN4       ;/
6 IN1       ;\pad1
7 IN3       ;/
8 CK1 (one short LOW pulse per JOY1/JOY3 data bit)
9 CK2 (one short LOW pulse per JOY2/JOY4 data bit)
10 STB (one short HIGH pulse at begin of transfer)
11 GND
```

The daughterboard contains diodes in the CK1, CK2, STB lines, effectively making them open-collector outputs (so the joypad may require pull-up resistors for that signals).

SNES Audio/Video Connector Pinouts

RF Out (TV Modulator)
Cinch with channel switch. Modulated video signal with mono-audio.

Multi Out

```
1   RGB - Red analog video out       _______________---_______________
2   RGB - Green analog video out    /  11    9     7     5     3     1  \
3   RGB - H/V sync out             |                                     |
4   RGB - Blue analog video out     \__12____10____8_____6_____4_____2__/
5   Ground (used for Video)
6   Ground (used for Audio)
7   S-Video Y (luminance) out
8   S-Video C (chroma) out
9   Video Composite out (Yellow Cinch)
10  +5V DC
11  Audio Left out      (White Cinch)
12  Audio Right out     (Red Cinch)
```

Pin 1,2,4: Red/Green/Blue (1V DC offset, 1V pp video into 75 ohms)
Pin 3,7,8,9: (1V pp into 75 ohms)
Pin 11,12: Left/Right (5V pp)
In cost-down SNES models, pin 1-4 and 7-8 are reportedly not connected (though one can upgrade them with some small modifications on the mainboard).

SNES Power Supply

AC 9V (PAL version)
Power Supply input. The 9V AC input is internally converted to 9V DC, and is then converted to 5V DC by a 7805 IC which turns the additional volts into heat. The CPU/PPU/APU are solely operated at 5V DC (ie. you can feed 5V to the 7805 output). However, 9V DC are used by the Audio Amplifier, the console works without the 9V supply, but you won't hear any sounds.
The Amplifier does work more less okay with 5V supply (ie. you can shortcut the 7805 input and output pins, and feed 5V to both of them) (when doing that disconnect the original 9V input to make sure that 9V aren't accidently passed to the CPU/PPU). However, at 5V, the middle amplitude levels are generated linearily intact, but the MIN/MAX levels are chopped off, for example a sawtooth wave looks like so:

```
Amplifier Output at 9V:              Amplifier Output at 5V:
   /|   /|   /|   /|   /| -8000h +2.5V     _    _    _    _    _
  / |  / |  / |  / |  / |                  / |  / |  / |  / |  / | -3XXXh +1V
 /  | /  | /  | /  | /  |                 /  |/  |/  |/  |/  |  | +3XXXh -1V
/   |/   |/   |/   |/   | +7FFFh -2.5V
```

The effect applies only if a game does use MIN/MAX levels (ie. there'd be no problem at Master Volume of 40h or less; unless additional output comes from Echo Unit).

10V DC 850mA (NTSC version)
Same as above, but using a DC supply (not AC), passed through a diode (so internally the voltage may drop from 10V to around 9V.

Power Switch / Anti-Eject
In older SNES consoles the Power switch comes with an anti-eject lever, which goes into a notch in the cartridge. Some carts have notches with square edges; cart cannot be removed while power is on. Other carts have notches with diagonal edge; the lever gets pushed (while sliding along the diagonal edge), and power is switched off automatically when removing the cartridge.
The anti-eject type is probably (?) used by carts with battery-backed SRAM, in order to prevent garbage writes (which might occur when simultaneously ejecting and powering-down).

SNES Expansion Port (EXT) Pinouts

EXT (at bottom of console) (used by Satellaview and Exertainment)
Most pins are exactly the same as on the cartridge slot, the only special pins, which aren't found on Cart slot, are EXT pins 21, 22, and 25: SMPCK is 8.192MHz (24.576MHz/3) from APU, DOTCK is the PPU dot clock (around 5.3MHz, with some stuttering during hblank period), MONO is a mono audio output.

```
          .--------.
       PA0 |1      2| PA1          Bottom View of console:
       PA2 |3      4| PA3       .-------------------------------------.
       PA4 |5      6| PA5       |              (rear side)            |
       PA6 |7      8| PA7       | +----+---------------+              |
     /PAWR |9     10| /PARD     | |snap|  28 ...... 2  |              |
        D0 |11    12| D1        | | in |  27 ...... 1  |              |
        D2 |13    14| D3        | +----+---------------+              |
        D4 |15    16| D5        |           EXT                       |
        D6 |17    18| D7        |                                     |
    /RESET |19    20| +5VDC     |                                     |
    SMPCLK |21    22| DOTCK     |                                     |
       GND |23    24| EXPAND    |                                     |
 MONO-AUDIO |25    26| /IRQ      |                                     |
   L-AUDIO |27    28| R-AUDIO   |              (front side)           |
          '--------'             '-------------------------------------'
```

The L/R-AUDIO inputs are essentially same as on cart slot, although they are passed through separate capacitors, so there is no 0 Ohm connection between EXT and Cart audio pins. EXT Pin 24 connects to Cart pin 2 (aside from a 10K pull-up it isn't connected anywhere inside of the console) so this pin is reserved for communication between cartridge hardware and ext hardware (used by Satellaview).

SNES Cartridge Slot Pinouts

Cartridge Slot
62 pins. Most cartridges are using only the middle 46 pins.

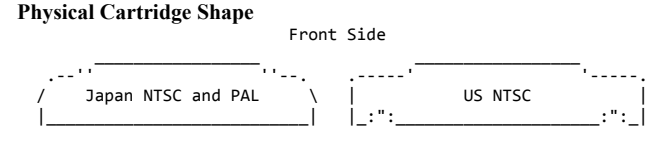
```
Front/Round    Rear/Flat
Solder side    Component side
MCK 21M - 01   32 - /WRAMSEL
EXPAND  - 02   33 - REFRESH
PA6     - 03   34 - PA7
/PARD   - 04   35 - /PAWR
           <key>
GND     - 05   36 - GND
A11     - 06   37 - A12
A10     - 07   38 - A13
A9      - 08   39 - A14
A8      - 09   40 - A15
A7      - 10   41 - A16
A6      - 11   42 - A17
A5      - 12   43 - A18
A4      - 13   44 - A19
A3      - 14   45 - A20
A2      - 15   46 - A21
A1      - 16   47 - A22
A0      - 17   48 - A23
/IRQ    - 18   49 - /ROMSEL
D0      - 19   50 - D4
D1      - 20   51 - D5
D2      - 21   52 - D6
D3      - 22   53 - D7
/RD     - 23   54 - /WR
CIC0    - 24   55 - CIC1
CIC2    - 25   56 - CIC3 3.072MHz
/RESET  - 26   57 - SYSCK
+5V     - 27   58 - +5V
           <key>
PA0     - 28   59 - PA1
PA2     - 29   60 - PA3
PA4     - 30   61 - PA5
SOUND-L - 31   62 - SOUND-R
          SHIELD
```

Caution: The connector uses a nonstandard 2.5mm pitch (not 2.54mm). And, the PCB is only 1.2mm thick (not 1.5mm).

Pin assignments

```
A23-0, D7-0, /WR, /RD - CPU address/data bus, read/write signals
/IRQ      - Interrupt Request (used by SA-1 and GSU)
/RESET    - When the system is reset (power-up or hard reset) this goes low
/WRAMSEL  - Work RAM select (00-3F,80-BF:0000-1FFF, 7E-7F:0000-FFFF)
/ROMSEL   - Cart ROM select (00-3F,80-BF:8000-FFFF, 40-7D,C0-FF:0000-FFFF)
PA7-0     - Address bus for $2100-$21FF range in banks $00-$3F/$80-$BF (B-Bus)
/PAWR     - Write strobe for B-Bus
/PARD     - Read strobe for B-Bus
MCK       - 21.47727 MHz master clock (used by SGB1 and MarioChip1)
SYSCK     - Unknown, is an output from the CPU.
SOUND-L/R - Left/Right Analog Audio Input, mixed with APU output (used by SGB)
EXPAND    - Connected to pin 24 of the EXT expansion port (for Satellaview)
REFRESH   - DRAM refresh (connects to WRAM, also used by SGB and SA-1)
              four HIGH pulses every 60us (every scanline)
              Used by SGB (maybe to sense SNES hblanks?)
CIC0      - Lockout Data to CIC chip in console    ;\from/to=initial direction
CIC1      - Lockout Data from CIC chip in console  ;/(on random-seed transfer)
CIC2      - Lockout Start (short HIGH pulse when releasing reset button)
CIC3      - Lockout Clock (3.072MHz) (24.576MHz/8) (from APU)
SHIELD    - GND (connected in SA-1 carts, SGB-also has provisions)
```

Physical Cartridge Shape



SNES Chipset

Chipset (PAL)

```
Board:     (C) 1992 Nintendo, SNSP-CPU-01        ;BOARD
U1  100pin Nintendo, S-CPU A, 5A22-02, 2FF 7S    ;CPU  (ID=2 in 4210h)
U2  100pin Nintendo, S-PPU1, 5C77-01, 2EU 64     ;PPU1 (ID=1 in 213Eh)
U3  100pin Nintendo, S-PPU2 B, 5C78-03, 2EV 7G   ;PPU2 (ID=3 in 213Fh)
U4  28pin  SONY JAPAN, CXK58257AM-12L, 227M87EY  ;VRAM1 32Kx8 SRAM
U5  28pin  SONY JAPAN, CXK58257AM-12L, 227M87EY  ;VRAM2 32Kx8 SRAM
U6  64pin  Nintendo, W-WRAM, 9227 T23 F          ;WRAM
U7  24pin  S-ENC, Nintendo, S (for Sony) 9226 B  ;video RGB to composite
U8  18pin  F413, (C) 1992, Nintendo, 9209 A      ;CIC
U9  -      N/A (NTSC version only, type 74HCU04) ;hex inverter (for X1 & CIC)
U10 14pin  (M)224, AN1324S (equivalent to LM324) ;SND Quad Amplifier
U11 3pin   T529D, 267                            ;GND,VCC,/RESET
U12 3pin   17805, 2F2, SV, JAPAN                 ;5V
U13 64pin  Nintendo, S-SMP, SONY, Nintendo'89... ;SND1 (SPC700 CPU)
U14 80pin  Nintendo, S-DSP, SONY'89, WWW149D4X   ;SND2 (sound chip)
U15 28pin  MCM51L832F12, (M) JAPAN RZZZZ9224     ;SND-RAM1 32Kx8 SRAM
U16 28pin  MCM51L832F12, (M) JAPAN RZZZZ9224     ;SND-RAM2 32Kx8 SRAM
U17 16pin  NEC, D6376, 9225CJ (ie. NEC uPD6376)  ;SND-Dual 16bit D/A
U18 14pin  S-CLK, 2FS 4A (for PAL only)          ;X1 to 21.2MHz and 4.43MHz
TC1 2pin   Red Trimmer                           ;X1-ADJUST
X1  2pin   D177F2                                ;CPU/PPU 17.7344750MHz (PAL)
X2  2pin   CSA, 24.57MX, Gm J                    ;SND 24.576MHz
F1  2pin   SOC, 1.5A                             ;FUSE (supply-input)
T1  4pin   TDK, ZJYS-2, t                        ;DUAL-LOOP (supply-input)
DB1 4pin   TOSHIBA, 4B1, 1B 2-E JAPAN            ;AC-DC (PAL/AC-version only)
L1  2pin   220 (22uH)                            ;LOOP (color clock to GND)
VR1 2pin   (M)ZNR, FK220, 26                     ;? (supply)
J1? 2pin   AC Input 9V                           ;AC-IN
J2  4pin   SNSP CCIR-EEC, A E210265, 250142A     ;RF-Unit (modulator)
SW  4pin   Reset Button (on board)
P1  64pin  Cartridge Slot
P2  11pin  To Front Panel (Controllers/Power LED)
P3  2pin   To Power Switch
P4  12pin  Multi Out
P5  28pin  EXT Expansion Port (bottom side)
```

Costdown SNES chipset

```
U1 160pin  Nintendo S-CPUN A, RF5A122 (CPU, PPU1, PPU2, S-CLK)
U2 100pin  Nintendo S-APU             (S-SMP, S-DSP, 64Kx8 Sound RAM)
U3  64pin  Nintendo S-WRAM B
U4  28pin  32Kx8 SRAM (video ram)
U5  28pin  32Kx8 SRAM (video ram)
U6?  8pin? ?
U7  24pin  Nintendo S-RGB A
U8  18pin  Nintendo F411B (CIC)
U9   3pin  17805 (5V supply)
U10 14pin  S-MIX A (maybe sound amplifier?)
U11  3pin  Reset?
X1   2pin  D21G8N (21.4MHz NTSC, or 17.7MHz PAL)
X2   2pin  APU clock (probably the usual 24.576MHz?)
```

51832

```
Toshiba TC51832FL-12 32k x8 SRAM (SOP28)
```

CXK58257AM-12L

```
32768-word x 8-bit high speed CMOS static RAM, 120ns,
standby 2.5uW in 28-pin SOP package.
Operational temperature range from 0'C to 70'C.
```

SNES Pinouts CPU Chip

CPU 5A22

```
  1     In  VCC (supply)
  2-17  Out A8..A23
  18    In  GND (supply)
  19-26 I/O JOY-IO-0..7 (Port 4201h/4213h.Bit0..7)
  27-31 In  JOY-2 (4017h.Read.Bit0..4) (Pin 29-31 wired to GND)
  32-33 In  JOY-1 (4016h.Read.Bit0..1)
  34    In   "VCC" (unknown, this is NOT 4016h.Bit2) (wired to VCC)
  35    Out JOY-1-CLK (strobed on 4016h.Read)
  36    Out JOY-2-CLK (strobed on 4017h.Read)
  37-39 Out JOY-OUT0..2 (4016h.Write.Bit0-2, OUT0=JOY-STROBE,OUT1-OUT2=UNUSED?)
  40    Out REFRESH (DRAM refresh for WRAM, four HIGH pulses per scanline)
  41-42 In  TCKSEL0,TCKSEL1 (wired to GND) (purpose unknown)
  43-44 In  HBLANK,VBLANK (from PPU, for H/V-timers and V-Blank NMI)
  45    In  /NMI (wired to VCC)
  46    In  /IRQ (wired to Cartridge and Expansion Port)
  47    In  GND (supply)
  48    In  MCK ;21.47727 MHz master clock ;measured ca.21.666666MHz? low volts
  49    In  /DRAMMODE (wired to GND) (allows to disable DRAM refresh)
  50    In  /RESET
  51-58 Out PA0..PA7
  59    In  VCC (supply)
  60-67 I/O D0-D7
  68    Out /PARD
  69    Out /PAWR
  70    Out /DMA (NC)    HI
  71    Out CPUCK (NC)      LOOKS SAME AS PIN72 (MAYBE PHASE-SHIFTED?)
  72    Out SYSCK (to WRAM and Cartridge) FAST CLK... TYPICALLY 2xHI, 1xLO
  73    In  TM (wired to GND) (purpose unknown)
  74    In  HVCMODE (wired to GND) (purpose unknown)
  75    In  HALT (wired to GND) (purpose unknown) (related to RDY?)
  76    In  /ABORT (wired to VCC)
  77    Out /ROMSEL (access to 00-3F/80-BF:8000-FFFF or 40-7D/C0-FF:0000-FFFF)
  78    Out /WRAMSEL (access to 00-3F/80-BF:0000-1FFF or 7E-7F:0000-FFFF)
  79    In  GND (supply)
  80    Out R/W (NC) (ie. almost same as /WR, but with longer LOW-duty)
  81    In  RDY (wired to VCC) (schematic says "PE" or RE" or so?)
  82    Out /ML (NC) (memory lock,low on read-modify,ie.inc/dec/shift/etc)
  83    Out MF (NC) (CPU's M-Flag, 8bit/16bit mode)
  84    Out XF (NC) (CPU's X-Flag, 8bit/16bit mode)
  85    In  VCC (supply)
  86    Out VFB or VPB or so (NC)    RAPID PULSED
  87    Out VFA or VPA or so (NC)    RAPID PULSED
  88    Out ALCK                     LOOKS LIKE INVERSE OF PIN71 or PIN72
  89    Out /VP (NC) (vector pull, low when executing exception vector)
  90    In  GND
  91    Out /WR (low on any memory write, including io-writes to 21xxh/4xxxh)
  92    Out /RD
 93-100 Out  A0..A7
```

The three VCC pins are interconnected inside of the chip (verified).
The various GND pins are not verified (some may be supply, or inputs).

SNES Pinouts PPU Chips

S-PPU1, 5C77

```
  1     ?      TST1 (GND)
  2     ?      TST0 (GND)
  3     CPU    /PARD
  4     CPU    /PAWR
  5-12  CPU    PA7-PA0 (main cpu b-bus)
  13    Supply VCC
  14-21 CPU    D7-D0 (main cpu)
  22    Supply GND
  23    ?      HVCMODE (GND)
  24    Mode   PALMODE (VCC=PAL or GND=NTSC)
  25    ?      /MASTER (GND)
  26    ?      /EXTSYNC (VCC)
  27    ?      NC (GND)
  28-35 SRAM   DH0-DH7 (sram data bus high-bytes)
  36    Supply VCC
  37-44 SRAM   DL0-DL7 (sram data bus low-bytes)
  45    Supply GND
  46    SRAM   VA15 (NC)  (would be for 64K-word RAM, SNES has only 32K-words)
  47    SRAM   VA14       (sram address bus for upper/lower 8bit data)
  48-61 SRAM   VAB13-VAB0 (sram address bus for upper 8bit data)
  62    Supply VCC
  63-76 VRAM   VAA13-VAA0 (sram address bus for lower 8bit data)
  77    Supply GND
  78    VRAM   /VAWR (sram write lower 8bit data)
  79    VRAM   /VBWR (sram write upper 8bit data)
  80    VRAM   /VRD  (sram read 16bit data)
  81    Supply VCC
  82-85 PPU    CHR3-CHR0      ;\
  86-87 PPU    PRIO1-PRIO0    ;
  88-90 PPU    COLOR2-COLOR0  ;/
  91           /VCLD (20ms high, 60us low)  (LOW during V=0)
  92           /HCLD (60us high, 0.2us low) (low during 11th dot-cycle
                                             of the 15-cycle color burst)
  93           /5MOUT (shortcut with pin 97, /5MIN) (and to PPU2, /5MIN)
                       8 clks = 7.5*0.2us = 5.333MHz
  94           /OVEP (always high?) (to PPU2 /OVER1 and /OVER2)
  95           FIELD (NTSC: 30Hz, PAL: 25Hz) (signalizes even/odd frame)
  96    Supply GND
  97           /5MIN (shortcut with pin 93, /5MOUT)  (as above 5mout)
  98    PPU    /RESET (from PPU2 /RESOUT0)
  99    ?      TST2 (GND)
  100   System XIN (21MHz)
```

S-PPU2, 5C78

```
2     ?      /PED (NC) (ca. 15kHz, hblank related, 50us high, 10us low)
3     Video  3.58M (to NTSC encoder)    5 clks = 1.4us
4     ?      /TOUMEI (NC) (LOW during V-Blank and H-Blank) (or vram access?)
5     Supply VCC
6     CPU    /PAWR
7     CPU    /PARD
8-15  CPU    D7-D0
16    Supply GND
17-24 CPU    PA7-PA0
25    CPU    HBLANK (for CPU h/v-timers)
              high during last some pixels, right border HSYNC,lead,burst
              low during last some burst clks, left border, and most pixels
26    CPU    VBLANK (for CPU h/v-timers)
              high during VBLANK (line 225-261)
              low during prepare (line 0) and picture (line 1-224)
27           /5MOUT (via 100 ohm to DOTCK on Expansion Port Pin22)
28    System /RESOUT1 (via 1K to CPU, APU, Cartridge, Expansion, etc.)
29    Joy    EXTLATCH (Lightpen signal)
30    Mode   PALMODE (VCC=PAL or GND=NTSC)
31    System XIN (21MHz)
32    Supply VCC
33    PPU    /RESOUT0 (to PPU1 /RESET)
34    CIC    /RESET (from CIC Lockout chip & Reset Button)
35    Supply GND
36           FIELD (NTSC: 30Hz, PAL: 25Hz) (signalizes even/odd frame)
37           /OVER1   ??
38           /5MIN (from PPU1)
39           /HCLD (low during 11th dot-cycle of the 15-cycle color burst)
40           /VCLD (LOW during V=0)
41-43 PPU    OBJ0-OBJ2 (COLOR0-COLOR2)
44-45 PPU    OBJ3-OBJ4 (PRIO0-PRIO1)
46-49 PPU    OBJ5-OBJ8 (CHR0-CHR3)
50           /OVER2
51-58 SRAM   VDB0-VDB7 (sram data upper 8bit)
59    Supply VCC
60-67 SRAM   VDA0-VDA7 (sram data lower 8bit)
68    Supply GND
69-76 SRAM   EXT0-EXT7 (sram data upper 8bit) (shortcut with VDB0-VDB7)
77-82 ?      TST0-TST5 (NC) (always low?)
83    Supply VCC
84-89 ?      TST6-TST11 (NC) (always low?)
90-93 ?      TST12-TST15 (GND)
94    Supply AVCC (VCC)
95-97 Video  R,G,B (Analog RGB Output)
98    ?      HVCMODE (GND)
99    Supply GND
100   Video  /CSYNC (normally LOW during Hsync, but inverted during Vsync)
```

CPUN-A (160pin chip with CPU and PPU1 and PPU2 in one chip)
Used in newer cost-down SNES consoles. Pinouts unknown.

SNES Pinouts APU Chips

S-DSP Pinouts (Sound Chip)

```
1     CK     DKD (NC)              (5 clks = 1.2us) 4.096MHz (24.576MHz/6)
2     CK     MXK or MYX or so (NC) (5 clks = 1.6us) 3.072MHz (24.576MHz/8)
3-5   CK     MX1-MX3 (NC) 1.024MHz (24.576MHz/24) (3pins: phase/duty shifted)
6-8   SRAM   MD2-MD0 (SRAM Data)
9-11  SRAM   MA0-MA2 (SRAM Address)
12    Supply GND
13-19 SRAM   MA3-MA7,MA12,MA14 (SRAM Address)
20    SRAM   MA15 (NC) (instead, upper/lower 32K selected via /CE1 and /CE0)
21    ?      DIP (NC) (always high?)
22-32 SRAM   MD3,MD4,MD5,MD6,MD7,/CE1,/CE0,MA10,/OE,MA11,MA9
33    Supply VCC
34-36 SRAM   MA8,MA13,/WE
37    ?      TF (GND)    ;\wiring TF and/or TK to VCC crashes the SPC700
38    ?      TK (GND)    ;/(ie. they seem to mess up CPUK clock or SRAM bus)
39    Audio  /MUTE (to/after amplifier)
40    CK     MCK (NC) 64000Hz  (24.576MHz/24/16)
41    CIC    SCLK     3.072MHz (24.576MHz/8) (via inverters to "CIC" chips)
42    Audio  BCK      1.536MHz (24.576/16)       BitClk    ;\to uPD6376
43    Audio  LRCK     32000Hz  (24.576MHz/16/48) StereoClk ; D/A converter
44    Audio  DATA Data Bits (8xZeroPadding+16xData)        ;/
45-46 Osc    XTAO,XTAI (24.576MHz)
47    System /RESET
48    SPC700 CPUK     2.048MHz (24.576MHz/12) (to S-SMP)
49    SPC700 PD2 (on boot: always high?)
50    SPC700 PD3 (on boot: always low?)
51    SPC700 D0
52    Supply GND
53-59 SPC700 D1-D7
60-72 SPC700 A0-A12
73    Supply VCC
74-76 SPC700 A13-A15
77    CK     XCK 24.576MHz (24.576MHz/1) (NC)
78    Exp.   DCK  8.192MHz (24.576MHz/3) (to Expansion Port Pin 21, SMPCLK)
79    CK     CK1 12.288MHz (24.576MHz/2) (NC)
80    CK     CK2  6.144MHz (24.576MHz/4) (NC)
```

S-SMP Pinouts (SPC700 CPU)

```
1-5   DSP    A4..A0 Address Bus
6-13  DSP    D7..D0 Data Bus
14    DSP    PD3 (maybe R/W signal or RAM/DSP select?)
15    DSP    PD2 (maybe R/W signal or RAM/DSP select?)
16    DSP    CPUK (2.048MHz from DSP chip)
17    AUX    /P5RD (NC)
18-25 AUX    P57..P50 (NC)
```

```
  27-34 AUX    P47..P40 (NC)
  35    ?      T1 (NC or wired to VCC) (maybe test or timer?)
  36    ?      T0 (NC or wired to VCC) (maybe test or timer?)
  37    System /RESET
  38-45 CPU    D7..D0 Data Bus
  46-51 CPU    /PARD,/PAWR,PA1,PA0,PA6 (aka CS), PA7 (aka /CS) B-Address Bus
  52-56 DSP    A15-A11 Address Bus
  57    Supply VCC (5V)
  58    Supply GND
  59-64 DSP    A10-A5 Address Bus
```

Pin 1-16 and 52-64 to S-DSP chip, Pin 17-34 Aux (not connected), Pin35-36 wired to VCC (schematic) actually NC (real hw), pin 37-51 to main CPU.

CPUK caution:

```
 scope measure with "x10" ref (gives the correct signal):
   -_-_-_-_-_-_-_-_ 2.048MHz (0.5us per cycle)
 during (and AFTER) "x1" ref (this seems to "crash" the clock generator):
   ---_---_---_---_ 1.024MHz (1.0us per cycle) (with triple-high duty)
```

S-APU Pinouts (S-SMP, S-DSP, 64Kx8 Sound RAM)

This 100pin chip is used in newer cost-down SNES consoles.

```
  1-100 Unknown
```

It combines the S-SMP, S-DSP, and the two 32Kx8 SRAMs in one chip. And, possibly also the NEC uPD6376 D/A converter?

SNES Pinouts ROM Chips

```
          _________  _________
    GND | 01       \_/       40 | VCC
    GND | 02 ......   .......39 | VCC                 ________  ________
--> A20 | 03 01    \./    36 38 | VCC            ?  | 01     \_/     36 | VCC
GND,A21 | 04 02 ...   ... 35 37 | A22,GND <--    ?  | 02 ....   .... 35 | ?
--> A17 | 05 03 01 \./ 32 34 36 | NC,VCC  <--   NC  | 03 01  \./  32 34 | VCC
--> A18 | 06 04 02     31 33 35 | /CS <--      A16  | 04 02       31 33 | NC
    A15 | 07 05 03     30 32 34 | A19 <--      A15  | 05 03       30 32 | A17
    A12 | 08 06 04     29 31 33 | A14          A12  | 06 04       29 31 | A14
     A7 | 09 07 05 ROM 28 30 32 | A13           A7  | 07 05 EPROM 28 30 | A13
     A6 | 10 08 06     27 29 31 | A8            A6  | 08 06 style 27 29 | A8
     A5 | 11 09 07     26 28 30 | A9            A5  | 09 07 (eg.  26 28 | A9
     A4 | 12 10 08     25 27 29 | A11           A4  | 10 08 in    25 27 | A11
     A3 | 13 11 09     24 26 28 | A16 <--       A3  | 11 09 SGB)  24 26 | /OE
     A2 | 14 12 10     23 25 27 | A10           A2  | 12 10       23 25 | A10
     A1 | 15 13 11     22 24 26 | /RD           A1  | 13 11       22 24 | /CS
     A0 | 16 14 12     21 23 25 | D7            A0  | 14 12       21 23 | D7
     D0 | 17 15 13     20 22 24 | D6            D0  | 15 13       20 22 | D6
     D1 | 18 16 14     19 21 23 | D5            D1  | 16 14       19 21 | D5
     D2 | 19 17 15     18 20 22 | D4            D2  | 17 15       18 20 | D4
    GND | 20 18 16     17 19 21 | D3           GND  | 18 16       17 19 | D3
        |_____________________ _|                   |____________________|
```

Note that /CS and A16..A22 are located elsewhere as on EPROMs.

More pins:

```
  S-DD1 is bundled with a 44pin ROM.
  SPC7110 with one or two 40pin ROMs.
```

Some ROMs (in SGB for example) have /CS and /OE arranged as in EPROMs.

```
      _____  _____
A22 |   1  \_/  44 | A21
A19 |   2       43 | A20
A18 |   3       42 | A9
A8  |   4       41 | A10
A7  |   5       40 | A11
A6  |   6       39 | A12
A5  |   7       38 | A13
A4  |   8       37 | A14
A3  |   9       36 | A15
A2  |  10       35 | A16
A1  |  11       34 | A17
/CE |  12       33 | BHE (HI)
GND |  13       32 | GND
/OE |  14       31 | D15,A0
D0  |  15       30 | D7
D8  |  16       29 | D14
D1  |  17       28 | D6
D9  |  18       27 | D13
D2  |  19       26 | D5
D10 |  20       25 | D12
D3  |  21       24 | D4
D11 |  22       23 | VCC
    |_______________|
```

44pin ROMs are used by (some) SPC7110 boards (with 8bit databus), and by (all) S-DD1 and SA-1 boards (existing photos look as if: with 16bit databus).

SNES Pinouts RAM Chips

128K WRAM Pinouts

```
  1 VCC      9  -            17 GND 25 A1  33 GND 41 A8      49 VCC    57 /RD
  2 D4       10 CS,VCC      18 -   26 A10 34 A13 42 ENA,A22 50 PS,PA7 58 /PAWR
  3 D5       11 CS,VCC      19 -   27 A2  35 A5  43 /PS,PA2 51 PS,VCC 59 /WR
  4 D6       12 CS,VCC      20 -   28 A11 36 A14 44 /PS,PA3 52 PS,VCC 60 D0
  5 D7       13 /CS,GND     21 -   29 A3  37 A6  45 /PS,PA4 53 PA0    61 D1
  6 SYSCK    14 /CS,GND     22 -   30 A12 38 A15 46 /PS,PA5 54 PA1    62 D2
  7 REFRESH  15 /CS,/WRAMSEL 23 A0  31 A4  39 A7  47 /PS,PA6 55 G (NC) 63 D3
  8 /RESET   16 VCC         24 A9  32 VCC 40 A16 48 GND     56 /PARD  64 GND
```

Note: The WRAM is Dynamic RAM (DRAM) and does require REFRESH pulses. If REFRESH is disabled (via DRAMMODE pin) then WRAM forgets its content after some minutes. Whereas, refresh occurs also on /RD (with the refresh row output on A0..A8), so, for example, games that are DMA'ing 512 bytes

from WRAM to OAM in every frame should work perfectly without REFRESH.
Interestingly, WRAM is kept intact even if the RESET button is held down for about 30 minutes (although the CPU doesn't generate any /RD, REFRESH, nor A0..A8 cycles during that time).

SRAM Pinouts

```
        .-------------__-------------.
     NC |1                         36| VCC
    A20 |2  ..........__.......... 35| A19
 NC,A18 |3  1                    32 34| NC,VCC (NC, ie. not CE2)
    A16 |4  2  ........__........ 31 33| A15
 NC,A14 |5  3  1              28 30 32| A17,CE2,VCC
    A12 |6  4  2  ....__.... 27 29 31| /WE
     A7 |7  5  3  1       24 26 28 30| A13,CE2,VCC
     A6 |8  6  4  2       23 25 27 29| A8
     A5 |9  7  5  3       22 24 26 28| A9
     A4 |10 8  6  4  SRAM 21 23 25 27| A11,/WE
     A3 |11 9  7  5       20 22 24 26| /OE
     A2 |12 10 8  6       19 21 23 25| A10
     A1 |13 11 9  7       18 20 22 24| /CE,/CE1
     A0 |14 12 10 8       17 19 21 23| D7
     D0 |15 13 11 9       16 18 20 22| D6
     D1 |16 14 12 10      15 17 19 21| D5
     D2 |17 15 13 11      14 16 18 20| D4
    GND |18 16 14 12      13 15 17 19| D3
        '----------------------------'
```

28pin 32Kbyte SRAM is used for Video RAM and Sound RAM (on mainboard)
Various SRAM sizes are used in game cartridges.

SNES Pinouts CIC Chips

CIC Pinouts
F411/F413: 18pin SMD-chip (used in console and in some carts)
D411/D413: 16pin DIP-chip (used in most carts)

SMD	DIP	Pin	Dir	Usage	In Console	In Cartridge
1	1	P00	Out	DTA0	Cart.55 CIC1	Cart.24 CIC0
2	2	P01	In	DTA1	Cart.24 CIC0	Cart.55 CIC1
3	3	P02	In	RANDOM	Via capacitor to VCC	NC
4	4	P03	In	MODE	VCC=Console (Lock)	GND=Cartridge (Key)
5		NC	-	(NC)	NC	NC
6	5	CL2	-	(NC)	NC	SMD:NC or DIP:GND
7	6	CL1	In	CLK	3.072MHz (from APU)	Cart.56 CIC3 (3.072MHz)
8	7	RES	In	RESET	From Reset button	Cart.25 CIC2 (START)
9	8	GND	-	GND	Supply	Supply
10	9	P10	Out	/RESET	To PPU (and CPU/APU/etc)	NC (or to ROM, eg. in SGB)
11	10	P11	Out	START	Cart.25 CIC2	NC
12	11	P12	-	(NC)	NC	NC (or SlotID in FamicomBox)
13	12	P13	-	(NC)	NC	NC (or SlotID in FamicomBox)
14		NC	-	(NC)	NC	NC
15	13	P20	-	(NC)	NC	NC
16	14	P21	-	(NC)	NC	NC (or SlotID in FamicomBox)
17	15	P22	-	(NC)	NC	NC (or SlotID in FamicomBox)
18	16	VCC	-	VCC	Supply	Supply

P00=Out,P01=In are the initial directions (for the Random Seed transfer), later on the directions are randomly swapped (ie. P00=In,P01=Out or P00=Out,P01=In).
START: short HIGH pulse on power-up or when releasing reset button.
/RESET: in console: to PPU, and from there to CPU,APU,Cart,Expansion.

SNES Pinouts MAD Chips

MAD-1 (and MAD-1 A) Pinouts (Memory Address Decoder 1)

```
 1  OUT1 /ROM.CS1 ;Chipselect to Upper ROM (NC if single ROM)
 2  OUT2 /SRAM.CS ;Chipselect to SRAM
 3  OUT3 /AUX.CS  ;Chipselect to Expansion I/O or so (usually NC)
 4  OUT4 /ROM.CS  ;Chipselect to Single ROM (NC if two ROMs)
 5  Vout          ;Supply to SRAM (+3V when VCC=off, +5V when VCC=on)
 6  VCC           ;Supply from SNES (+5V)
 7  Vbat          ;Supply from Battery via resistor (+3V)
 8  GND           ;Supply Ground
 9  IN6  /RESET   ;From cart.26
10 IN5  MODE     ;HiROM: VCC                 | LoROM: GND
11 IN4  /ROMSEL  ;From cart.49
12 IN3  Addr3    ;HiROM: A22 (400000h) or A15 | LoROM: A22 (400000h) or VCC
13 IN2  Addr2    ;HiROM: A21 (200000h)        | LoROM: A21 (200000h)
14 IN1  Addr1    ;HiROM: A14 (4000h)          | LoROM: A20 (100000h)
15 IN0  Addr0    ;HiROM: A13 (2000h)          | LoROM: A15 (8000h)
16 OUT0 /ROM.CS0 ;Chipselect to Lower ROM (NC if single ROM)
```

Note that Addr3 is sometimes wired this or that way. And, when using two ROMs, Addr2 is used as upper ROM address line (so the ROM-size may affect the SRAM/AUX mapping).

MAD-1 (and MAD-1 A) Logic Table

```
 IN0   IN1   IN2   IN3   IN4   IN5   IN6   --> Output being
 Addr0 Addr1 Addr2 Addr3 /ROM  MODE  /RES      dragged LOW
 ------------------------------------------------------------
 HIGH  x     x     x     LOW   LOW   HIGH  --> /ROM.CS=LOW   ;\
 HIGH  x     LOW   x     LOW   LOW   HIGH  --> /ROM.CS0=LOW  ;
 HIGH  x     HIGH  x     LOW   LOW   HIGH  --> /ROM.CS1=LOW  ; LoROM
 LOW   LOW   HIGH  HIGH  LOW   LOW   HIGH  --> /AUX.CS=LOW   ;
 LOW   HIGH  HIGH  HIGH  LOW   LOW   HIGH  --> /SRAM.CS=LOW  ;/
 x     x     x     x     LOW   HIGH  HIGH  --> /ROM.CS=LOW   ;\
 x     x     LOW   x     LOW   HIGH  HIGH  --> /ROM.CS0=LOW  ;
 x     x     HIGH  x     LOW   HIGH  HIGH  --> /ROM.CS1=LOW  ; HiROM
 HIGH  HIGH  LOW   LOW   HIGH  HIGH  HIGH  --> /AUX.CS=LOW   ;
 HIGH  HIGH  HIGH  LOW   HIGH  HIGH  HIGH  --> /SRAM.CS=LOW  ;/
```

Unknown. Used in some newer DSPn cartridges (mainly/only DSP1B ones?) (probably similar to MAD-1, but maybe with some pins replaced by a clock amplifier for the DSPn oscillator... and/or maybe it supplies an inverted RESET signal to the DSPn chip).

MAD-R Pinouts (Memory Address Decoder with Reset-Inverter)
Pinouts are same as MAD-1, except for one pin: Pin4 outputs RESET (inverse of /RESET input) (there aren't any known cartridges using this feature).
The SHVC-2A3M-01, SHVC-2A3M-10, SHVC-2J3M-01 boards can be fitted with either MAD-1 or MAD-R (later SHVC-2A3M-11, SHVC-2J3M-11, SHVC-2J3M-20 boards are used with MAD-1 only, so MAD-R appears to have been discontinued after soon).
```
XXX The following boards (also) accept either MAD-1 or MAD-R:
2A1M-01
2A5M-01
2J5M-01
```

MAD-R Logic Table
```
IN0   IN1   IN2   IN3   IN4   IN5   IN6   --> Output being
Addr0 Addr1 Addr2 Addr3 /ROM  MODE  /RES      dragged LOW
-------------------------------------------------------------
x     x     x     x     x     x     HIGH  --> RESET=LOW    * ;-Reset
x     x     LOW   LOW   LOW   LOW   HIGH  --> /ROM.CS0=LOW * ;\
x     x     HIGH  LOW   LOW   LOW   HIGH  --> /ROM.CS1=LOW * ; LoROM
x     x     LOW   HIGH  LOW   LOW   HIGH  --> /AUX.CS=LOW  * ;
LOW   HIGH  HIGH  HIGH  LOW   LOW   HIGH  --> /SRAM.CS=LOW   ;/
x     x     LOW   x     LOW   HIGH  HIGH  --> /ROM.CS0=LOW   ;\
x     x     HIGH  x     LOW   HIGH  HIGH  --> /ROM.CS1=LOW   ; HiROM
HIGH  HIGH  LOW   LOW   HIGH  HIGH  HIGH  --> /AUX.CS=LOW    ;
HIGH  HIGH  HIGH  LOW   HIGH  HIGH  HIGH  --> /SRAM.CS=LOW   ;/
```
The four "*" marked lines are different as MAD-1.

Other Battery Controllers
Battery Controllers are used to write-protect the SRAM during power-up/power-down, and to switch from Vbat to VCC supply when available. Early carts used a transistor/diode circuit, later carts are using MAD-1/MAD-2/MAD-R or MM1026/MM1134 chips.

MM1026/MM1134 Pinouts
```
1 GND
2 /RESET (output)
3 CS (to SRAM) (usually NC when using /CS)
4 Vbat (from battery)
5 /CS (to SRAM)
6 Vout (to SRAM)
7 NC (MM1026) or /Y (MM1134)
8 VCC (from snes)
```
On MM1026, /CS is simply inverse of CS (both true when VCC=good). On MM1134, the additional /Y input allows to force /CS false (but doesn't affect CS).
On SA-1 "SNSP-1L3B-20" boards, the PCB text layer says MM1026AF, but the actual chip is labelled "6129A, 6C33"; which refers to a "BA6129A" chip.

Batteries
```
CR2032 (3 volt lithium cells, 20mm diameter, 3.2mm width) <-- most SNES carts
CR2430 (3 volt lithium cells, 24.5mm diameter, 3mm width) <-- X-Band Modem
NiCd (3.6V, rechargeable, but acid-leaking) <-- many Copiers
```

MAD Versions/Revisions & Power Down Notes
```
BU2230           MAD-1
XLU2230          MAD-1
BU2230A          MAD-1A
BU2231A          MAD-2A
BU2220           MAD-R
TexasInstruments MAD-1
```
At VCC>Vbat, the chips do operate normally (as shown in above Logic Tables).
At VCC=Vbat, the BU2230,BU2230A,XLU2230 (MAD1/MAD1A) force /SRAM.CS=high, whilst TexasInstrument (MAD1) and BU2220 (MAD-R) keep /SRAM.CS operating as normally.
At VCC=0, the BU2230,BU2230A,XLU2230 (MAD1/MAD1A) chips and BU2220 (MAD-R) switch the other 4 outputs to LOW, whilst TexasInstrument (MAD1) switch them HIGH (or maybe they are left floating and get pulled high by the test circuit).

SNES Pinouts RTC Chips

Sharp S-RTC Pin-Outs (used by Dai Kaiju Monogatari 2)
```
1-24 Unknown (should have an address decoder and 4bit data bus or so)
```
24pin chip. Still unknown which & how many address/data lines are connected, and if there are "specials" like /IRQs (?)

Epson/Seiko RTC-4513 Pin-Outs (for Far East of Eden Zero) (via SPC7110 chip)
```
1 NC
2 DATA
3 STD.P
4 NC
5 NC
6 VCC
7 NC
8 NC
9 GND
10 NC
11 NC
12 CE
13 CLK
14 NC
```

Seiko/Epson S-3520CF Pin-Outs (used in SFC-Box and NSS)
```
1 Xin
2 NC
3 Xout
4 /CLK
5 DataIn
6 /WR
7 GND
8 /TPOUT
9 DataOut
```

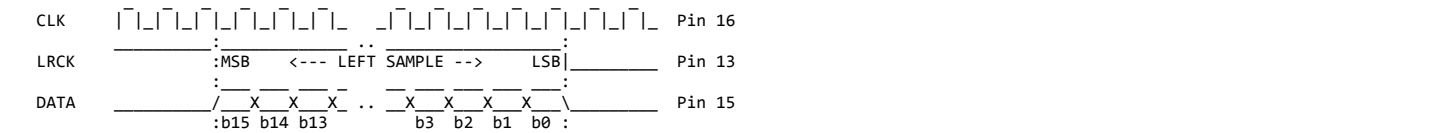
```
10 PDW
11 /CS
12 Capacitor
13 NC
14 VCC
```
Crystal = 32.768kHz (see datasheet page 13)

SNES Pinouts Misc Chips

S-ENC Pin-Outs
```
1 (R-Y)O     5 VCC    9 YI        13 VC (clk)   17 PHA (NC)    21 AG (Green)
2 GND        6 CO     10 (B-Y)I   14 VB (clk)   18 PDO (NC)    22 AB (Blue)
3 PCP,/BURST 7 VO     11 (R-Y)I   15 VA (NC)    19 NTSC/PAL    23 YO
4 SW (VCC)   8 SYNC   12 BLA      16 BFP,/BURST 20 AR (Red)    24 (B-Y)O
```
Analog RGB to composite converter. Pin19 allows to select PAL or NTSC mode (also requires the correct PAL or NTSC clock input on Pin13,14).
The "S-RGB A" has reportedly other pinouts:
```
1 ?          5 ?      9 ?           13 ?        17 Y (luma)    21 ?
2 ?          6 ?      10 ?          14 ?        18 ?           22 Green
3 ?          7 CSYNC  11 ?          15 ?        19 ?           23 ?
4 ?          8 ?      12 C (chroma) 16 ?        20 Red         24 Blue
```

S-CLK Pin-Outs (PAL only)
```
1 17.7MHz(X1.A)   4 21.28MHz(MCK)   7 3.072MHz(Cart)  10 GND    13 Low
2 17.7MHz(X1.B)   5 4.433MHz(PAL)   8 3.072MHz(APU)   11 Low    14 VCC
3 VCC             6 3.072MHz(CIC)   9 3.072MHz(APU)   12 Low
```
Clock multiplier/divider for PAL consolses (none such in NTSC consoles).
Pin6-9 are two inverters (for APU-generated CIC clock) (NTSC consoles have equivalent inverters in a 74HCU04 chip).

NEC uPD6376 (two-channel serial 16bit D/A Converter)
```
1 DSSEL (GND)    5 AGND (NC)     9 RREF           13 LRCK (32000Hz)
2 DGND (GND)     6 ROUT          10 LREF          14 LRSEL (GND)
3 NC (VCC)       7 AVDD (VCC)    11 LOUT          15 SI (DATA)
4 DVDD (VCC)     8 AVDD (VCC)    12 AGND (GND)    16 CLK (1.536MHz)
```
DATA changes on falling CLK and is valid on raising CLK, falling LRCK indicates that the most recent 16 bits were LEFT data, raising LRCK indicates RIGHT data. Each 16bit sample is preceeded by 8 dummy bits (which are always zero). The 16bit values are signed without BIAS offset (0000h=silence).

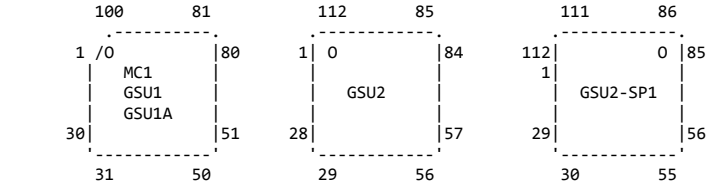


If the MUTE bit is set in FLG.6, then DATA is always 0, and additionally /MUTE=LOW is output to the Amplifier (so the DSP is "double-muted", and, the /MUTE signal also mutes external sound inputs like from SGB).

S-MIX Pin-Outs
Unnown. This chip is found on some cost-down SNES mainboards. Maybe a sound amplifier.

LM324 Quad Amplifier
```
1 LeftPostOut      8 LeftPreOut
2 LeftPostIn-      9 LeftPreIn-
3 LeftPostIn+      10 LeftPreIn+
4 VS (not VCC)     11 GND
5 RightPostIn+     12 RightPreIn+
6 RightPostIn-     13 RightPreIn-
7 RightPostOut     14 RightPreOut
```
The Pre-amplifiers do amplify the signal from the D/A converter. The pre-amplified signal is then mixed (via resisters and capacitors) with the other two audio sources (from cartridge and expansion port), the result is then passed through the Post-amplifier stage, eventually muted via transistors (if DSP outputs /MUTE=Low), and amplified through further transistors. The final stereo signal is output to A/V connector, and mono signals are mixed (via resistors) for Expansion Port and TV Modulator). The LM324 chip and the transistors are using the VS supply, rather than normal 5V VCC.

SNES Pinouts GSU Chips

GSU Chip Packages


GSU2-SP1 is having odd pin numbering (with pin1 being the SECOND pin; which was apparently done to maintain same pin numbers as for GSU2).

MC1, GSU1, and GSU1A
```
 1 GND
 2 ROM.A18
 3 ROM.A17
 4 ROM.A16
 5 ROM.A15
 6 ROM.A14
 7 ROM.A13
 8 ROM.A12
 9 ROM.A11
10 ROM.A10
11 ROM.A9
12 ROM.A8
13 ROM.A7
14 ROM.A6
```

```
 16 ROM.A4
 17 ROM.A3
 18 ROM.A2
 19 ROM.A1
 20 ROM.A0
 21 ROM.D7
 22 ROM.D6
 23 ROM.D5
 24 ROM.D4
 25 ROM.D3
 26 ROM.D2
 27 GND
 28 ROM.D1
 29 ROM.D0
 30 VCC
 --
 31 ?
 32 /WR
 33 /RD
 34 /RESET
 35 D7
 36 D6
 37 D5
 38 D4
 39 D3
 40 GND    ;\swapped on GSU2
 41 VCC    ;/
 42 D2
 43 D1
 44 D0
 45 A22
 46 A21
 47 A20
 48 A19
 49 A18
 50 A17
 --
 51 A16
 52 A15
 53 A14
 54 A13
 55 A12
 56 /IRQ
 57 A0
 58 A1
 59 A2
 60 A3
 61 A4
 62 A5
 63 A6
 64 A7
 65 A8
 66 A9
 67 A10
 68 A11
 69 GND
 70 X1 (21.44MHz ?)
 71 SRAM.D0
 72 SRAM.D1
 73 SRAM.D2
 74 SRAM.D3
 75 SRAM.D4
 76 SRAM.D5
 77 SRAM.D6
 78 SRAM.D7
 79 SRAM.A0
 80 SRAM.A1
 --
 81 SRAM.A2
 82 SRAM.A3
 83 SRAM.A4
 84 SRAM.A5
 85 SRAM.A6
 86 SRAM.A7
 87 SRAM.A8
 88 SRAM.A9
 89 VCC
 90 GND
 91 SRAM.A10
 92 SRAM.A11
 93 SRAM.A12
 94 SRAM.A13
 95 SRAM.A14
 96 GND
 97 SRAM.A15
 98 SRAM./OE
 99 SRAM./WE
100 ROM.A19
```

GSU2, and GSU2-SP1

```
  1 ROM.A17
  2 ROM.A16
  3 ROM.A15
  4 ROM.A14
  5 ROM.A13
  6 ROM.A12
  7 ROM.A11
  8 ROM.A10
  9 ROM.A9
 10 ROM.A8
 11 ROM.A7
```

```
13 ROM.A5
14 VCC
15 ROM.A4
16 ROM.A3
17 ROM.A2
18 ROM.A1
19 ROM.A0
20 ROM./CE
21 ?       (NC, probably /CE for 2nd ROM chip)
22 ROM.D7
23 ROM.D6
24 ROM.D5
25 ROM.D4
26 ROM.D3
27 ROM.D2
28 GND?
--
29 ROM.D1
30 ROM.D0
31 ?
32 ?
33 /WR
34 /RD
35 /RESET
36 GND?
37 D7
38 D6
39 D5
40 D4
41 D3
42 VCC
43 GND
44 D2
45 D1
46 D0
47 A23
48 A22
49 A21
50 A20
51 A19
52 A18
53 A17
54 A16
55 A15
56 A14
--
57 A13
58 A12
59 /IRQ
60 A0
61 A1
62 A2
63 A3
64 A4
65 GND?
66 A5
67 A6
68 A7
69 A8
70 VCC
71 A9
72 A10
73 A11
74 GND
75 X1 (21.44MHz)
76 VCC
77 SRAM.D0
78 SRAM.D1
79 SRAM.D2
80 SRAM.D3
81 SRAM.D4
82 SRAM.D5
83 SRAM.D6
84 SRAM.D7
--
85 SRAM.A0
86 SRAM.A1
87 SRAM.A2
88 SRAM.A3
89 SRAM.A4
90 SRAM.A5
91 SRAM.A6
92 SRAM.A7
93 SRAM.A8
94 SRAM.A9
95 NC?
96 NC?
97 VCC
98 VCC
99 GND
100 SRAM.A10
101 SRAM.A11
102 SRAM.A12
103 SRAM.A13
104 SRAM.A14
105 NC/SRAM.A15
106 NC/SRAM.A16
107 SRAM./OE
108 SRAM./WE
109 ROM.A20
110 ROM.A19
111 GND
```

```
112 ROM.A18
```

SNES Pinouts CX4 Chip

Capcom CX4 (used in Mega Man X2/X3)

```
  1  A3     21 A15    41 RA8    61 ???
  2  A4     22 A14    42 RA7    62 D7
  3  A5     23 A13    43 RA6    63 D6
  4  A6     24 A12    44 RA5    64 D5
  5  A7     25 ???    45 RA4    65 D4
  6  A8     26 /RCE1  46 RA3    66 Vcc
  7  A9     27 /RCE2  47 RA2    67 D3
  8  A10    28 RA19   48 RA1    68 D2
  9  A11    29 RA18   49 RA0    69 D1
  10 GND    30 RA17   50 GND    70 D0
  11 XIN    31 Vcc    51 ???    71 Vcc
  12 XOUT   32 RA16   52 /ROE   72 /RST
  13 A23    33 RA15   53 RD7    73 GND
  14 A22    34 RA20   54 RD6    74 GND
  15 A21    35 RA14   55 RD5    75 GNDed
  16 A20    36 RA13   56 RD4    76 /RD
  17 A19    37 RA12   57 RD3    77 /WR
  18 A18    38 RA11   58 RD2    78 A0
  19 A17    39 RA10   59 RD1    79 A1
  20 A16    40 RA9    60 RD0    80 A2
```

SNES bus (cartridge slot) connects to Pin 1-24 and 62-80, CX4 bus (ROM) to pin 26-60. Pin 25,51 are probably SRAM /CS and /WR (not used on existing CX4 boards). Pin 61 is unknown. Pin 75 is GNDed, but can be reconfigured [on some PCBs] via CL and R4 options (maybe for HiROM mapping).

SNES Pinouts SA1 Chip

SA-1

```
  1 SNES./IRQ
  2 SNES.D7
  3 SNES.D3
  4 SNES.D6
  5 SNES.D2
  6 SNES.D5
  7 SNES.D1
  8 SNES.D4
  9 SNES.D0
  10 VCC
  11 GND
  12 SNES.A23
  13 SNES.A0
  14 SNES.A22
  15 SNES.A1
  16 SNES.A21
  17 SNES.A2
  18 SNES.A20
  19 SNES.A3
  20 SNES.A19
  21 SNES.A4
  22 SNES.A18
  23 SNES.A5
  24 SNES.A17
  25 SNES.A6
  26 SNES.A16
  27 SNES.A7
  28 SNES.A15
  29 SNES.A8
  30 SNES.A14
  31 SNES.A9
  32 SNES.A13
  33 SNES.A10
  34 SNES.A12
  35 SNES.A11
  36 VCC
  37 GND
  38 REFRESH
 ---
  39 GND
  40 X.?     MasterClock (21.477MHz)
  41 X.?     MasterClock (21.477MHz)
  42 GND
  43 ROM.D15 pin31 (D15/A0)
  44 ROM.D7  pin30
  45 ROM.D14 pin29
  46 ROM.D6  pin28
  47 ROM.D11 pin22
  48 ROM.D3  pin21
  49 ROM.D10 pin20
  50 ROM.D2  pin19
  51 ROM.D13 pin27
  52 ROM.D5  pin26
  53 ROM.D12 pin25
  54 ROM.D4  pin24
  55 ROM.D9  pin18
  56 ROM.D1  pin17
  57 ROM.D8  pin16
  58 ROM.D0  pin15
  59 ROM.A1  pin11
  60 ROM.A2  pin10
  61 ROM.A3  pin9
  62 ROM.A4  pin8
  63 ROM.A5  pin7
  64 ROM.A6  pin6
```

```
  65 ROM.A7  pin5
  66 ROM.A8  pin4
  67 ROM.A9  pin42
  68 ROM.A10 pin41
  69 ROM.A11 pin40
  70 ROM.A12 pin39
  71 ROM.A13 pin38
  72 ROM.A14 pin37
  73 ROM.A15 pin36
  74 ROM.A16 pin35
  75 ROM.A17 pin34
  76 ROM.A19 pin2
  77 ROM.A18 pin3
  78 ROM.A20 pin43
  79 ROM.A21 pin44
  80 ROM.A22 pin1
  81           maybe A23 ?
  82 GND?
  83 VCC
  84 GND
  85 GND?
  86 SRAM. A16?  pin1-1 (extra pin)
  87 SRAM. A14   pin1
  88 SRAM. A12   pin2
  89 SRAM.A7  pin3
  90 SRAM.A6  pin4
  91 SRAM.A5  pin5
  92 SRAM.A4  pin6
  93 SRAM.A3  pin7
  94 SRAM.A2  pin8
  95 SRAM.A1  pin9
  96 SRAM.A0  pin10
  97 SRAM. A10   pin21
  98 SRAM. A11   pin23
  99 SRAM. A9    pin24
  100 GND
  101 VCC
  102 SRAM. A8   pin25
 ---
  103 to left-solder pads (U4.3.CS) (aka SRAM.A13)
  104 SRAM. A18?? pin1-2 (extra pin)
  105 SRAM. A15  pin28+1 (extra pin)
  106
  107
  108 SRAM. /OE  pin22
  109 SRAM. /WE  pin27
  110 SRAM.D0 pin11
  111 SRAM.D1 pin12
  112 SRAM.D2 pin13
  113 SRAM.D3 pin15
  114 SRAM.D4 pin16
  115 SRAM.D5 pin17
  116 SRAM.D6 pin18
  117 SRAM.D7 pin19
  118 GND
  119 VCC
  120 SNES./RESET
  121 SNES.SYSCK
  122 SNES.CIC3 (3.072MHz)
  123 SNES.CIC2
  124 SNES.CIC1
  125 SNES.CIC0
  126 SNES./WR
  127 PAL/NTSC (GND=NTSC, VCC=PAL) (for CIC mode and/or HV-timer?)
  128 SNES./RD
```

ROM-Chip Note: ROM./CE and ROM./OE are wired to GND (always enabled).
ROM-Chip Note: ROM.BHE is wired to VCC (always 16bit databus mode).

```
  U4.Pin5./CS ---> SRAM./CS   pin20  (U4:6129A aka PCB:MM1026AF)
  U4.Pin3.CS  ---> SRAM.A13?  pin26
  (left 4 solder-pads near U4 --> SRAM.pin26 = CS or A14)
  (right 4 solder-pads near U4 --> SRAM.pin28 = CS or Vbat)
```

Cart Slot Unused: /ROMSEL
Cart Slot Used: SHIELD (!)

SNES Pinouts Decompression Chips

SPC7110F0A Pin-Outs

```
  1 SnsA8   16 SnsA15   31 DatD7   46 GND     61 DatA9  76 SramCE2  91 GND
  2 GND     17 SnsA16   32 DatD6   47 Prg/CE  62 DatA8  77 RtcData  92 SnsD3
  3 SnsA7   18 SnsA17   33 DatD5   48 Dat/CE  63 VCC    78 RtcClk   93 SnsD2
  4 SnsA6   19 SnsA18   34 DatD4   49 DatA18  64 GND    79 Rtc/CE   94 SnsD1
  5 SnsA5   20 SnsS19   35 GND     50 DatA17  65 DatA7  80 VCC      95 SnsD0
  6 SnsA4   21 SnsA20   36 DatD3   51 VCC     66 DatA6  81 VCC      96 GND
  7 SnsA3   22 SnsA21   37 DatD2   52 GND     67 DatA5  82 GND      97 VCC
  8 SnsA2   23 SnsA22   38 DatD1   53 DatA16  68 DatA4  83 GND      98 SnsA11
  9 SnsA1   24 SnsA23   39 DatD0   54 DatA15  69 GND    84 GND      99 SnsA10
  10 SnsA0  25 Sns/RD   40 VCC     55 DatA14  70 DatA3  85 VCC      100 SnsA9
  11 GND    26 Sns/WR   41 GND     56 DatA13  71 DatA2  86 GND
  12 VCC    27 SnsRESET 42 DatA22  57 DatA12  72 DatA1  87 SnsD7
  13 SnsA12 28 Sns21MHz 43 DatA21  58 GND     73 DatA0  88 SnsD6
  14 SnsA13 29 Sns21MHz 44 DatA20  59 DatA11  74 VCC    89 SnsD5
  15 SnsA14 30 GND      45 DatA19  60 DatA10  75 GND    90 SnsD4
```

Sns=SNES Cart-Edge, Prg=Program ROM (/CE), Dat=Data ROM, Rtc=RTC, Sram=SRAM.

S-DD1 Pin-Outs

```
  Pin 1..100 = unknown
  Pin 82 is possibly CIC mode (PAL/NTSC mode)
```

SNES Pinouts BSX Connectors

FLASH Card Slot (as found on a "BSC-1A5M-01" board)
There are two conflicting numbering schemes for the 62pin connector.
Pin-numbering on the black plastic connector:
```
Rear/Left  --> 62 ............................. 32 <-- Rear/Right
Front/Left --> 31 ............................. 1  <-- Front/Right
```
Pin-numbering on SNES cartridge PCB text layer:
```
Rear/Left  --> 62 ............................. 2  <-- Rear/Right
Front/Left --> 61 ............................. 1  <-- Front/Right
```
Below is using the PCB text layer's numbering scheme:
```
1 GND
2 GND
3 D0
4 D4 (with cap to gnd)
5 D1 (with cap to gnd)
6 D5
7 D2
8 D6
9 D3
10 D7
11 A12
12 -
13 A7
14 via R2 to /RD (33 ohm)
15 A6
16 via R3 to /WR (33 ohm)
17 A5
18 VCC
19 A4
20 -
21 A3
22 via R4 to VCC (47kOhm)
23 A2
24 via R5 to GND (47kOhm)
25 A1
26 via R6 to GND (47kOhm)
27 A0
28 -
29 A14
30 VCC
31 VCC
32 VCC
33 via R7 to VCC (47kOhm)
34 GND
35 A13
36 REFRESH    to SNES.pin.33
37 A8
38 A15 rom     SNES.A16 SNES.pin.41
39 A9
40 A16 rom     SNES.A17 SNES.pin.42
41 A11
42 A17 rom     SNES.A18 SNES.pin.43
43 A10
44 A18 rom     SNES.A19 SNES.pin.44
45 SYSCK       SNES.pin57 (and via R1 to SNES.pin.2 EXPAND) (100 ohm)
46 A19 rom     SNES.A20 SNES.pin.45
47 /RESET
48 A20 rom     SNES.A21 SNES.pin.46
49 -
50 A21 rom     SNES.A23 SNES.pin.48 (NOT SNES.A22 !!!)
51 /CS (from MAD-1A.pin1)
52 GND
53 Dx
54 Dx
55 Dx
56 Dx      ... pins here are D8-D15 (on PCBs with 16bit databus)
57 Dx
58 Dx
59 Dx
60 Dx
61 GND
62 GND
```
pitch: 38.1mm per 30 pins === 1.27mm per pin
There are some connection variants: The Itoi cartridge with SA-1 is using 16bit databus (with extra data lines somewhere on pin53-60), pin12 seems to be connected to something, some of the the pull-up/pull-downs and VCC/GND pins on pin 14-34 and 52 may be wired differently.

SNES Cartridge Slot Usage for Satellaview BIOS and Datapack carts
REFRESH (SNES.pin33) is forwarded to FLASH cart slot (for unknown reason, maybe it is ACTUALLY used for deselecting FLASH during REFRESH, or maybe it was INTENDED for unreleased DRAM cartridges).
SYSCK (SNES.pin57) is forwarded to to FLASH cart slot (for unknown reason), and is also forwarded (via 100 ohms) to EXPAND (SNES.pin2) (and from there forwarded to the BSX Receiver Unit on SNES Expansion port).

BSX-EXT-Port Pinouts (half-way rev-engineered by byuu)
```
1  = +5V
2  = +5V
3  = +5V
4  = +5V
5  = GND
6  = GND
7  = GND
8  = GND
9  = GND
10 = GND
11 = U3.pin17 (B2) ;\
```

```
  13 = U3.pin15 (B4) ;
  14 = U3.pin16 (B3) ;
  15 = U3.pin13 (B6) ;
  16 = U3.pin14 (B5) ;
  17 = U3.pin11 (B8) ;
  18 = U3.pin12 (B7) ;/
  19 = U2.pin11 (Y8)
  20 = GND
  21 = U2.pin12 (Y7)
  22 = GND
  23 = ???
  24 = GND
  25 = U1.pin12 (Y7)
  26 = GND
  27 = U1.pin11 (Y8)
  28 = U1.pin13 (Y6)
  29 = ???
  30 = ???
  31 = ???
  32 = U2.pin14 (Y5)
  33 = ???
  34 = ???
  35 = GND
  36 = GND
  37 = GND
  38 = GND
```

The U3 transceiver is probably passing databus to/from SNES, the U1/U2 drivers are maybe passing some address/control signals from SNES. The indirect connection via U1/U2/U3 may be intended to amplify the bus, or to disconnect the bus (in case when the receiver power supply isn't connected).

SNES Common Mods

CIC Disable

The console contains a F411 (NTSC) or F413 (PAL) chip that verifies if the cartridge contains an identical chip, if it doesn't, then it resets the SNES, preventing to use unlicensed carts, or to use NTSC carts on PAL consoles.

```
  F411/F413 Pin 4 (GND=Disable/Unlock, VCC=Enable/Lock)
```

Even when disabled, some newer games (eg. Donkey Kong Country) may verify the PAL/NTSC framerate by software and refuse to run if it doesn't match the expected setting, this can be solved by adding a framerate switch (see below), the verification is often done only after power-up, so one can restore the desired setting after power-up.

Some newer games are reportedly also refusing to run if the CIC chip in the console is disabled, as a workaround, one would usually add a switch that allows to re-enables the CIC when needed. Eventually one could also modify the cartridges (they are probably connecting the CIC /RESET output to ROM CE2 pin or so?). Games with SA-1 or S-DD1 chips won't work.

50Hz/60Hz Switch

```
  PPU1 Pin 24 (GND=60Hz, VCC=50Hz)
  PPU2 Pin 30 (GND=60Hz, VCC=50Hz)
```

50Hz/60Hz Switch on newer cost-down SNES (those with 160pin S-CPUN A)

Basically, the frame rate is selected by a single pin:

```
  S-CPUN A, Pin 111 - PAL/NTSC  (high=PAL, low=NTSC)
```

An unwanted side effect is that this pin also changes the expected clock input:

```
  X1 oscillator (21.47727MHz=NTSC, 17.7344750MHz=PAL)
```

as a workaround, buy the missing oscillator, and use a "stereo" switch that simultaneously toggles the oscillator and the PAL/NTSC pin.

Another unwanted side effect is that it does (probably) change the color clock output for the S-RGB A chip, making the composite video signal unusable. As a workaround, one could use a TV set with RGB input (this would also require to connect the R,G,B,SYNC pins, which are left unconnected on the Multi-Out port of the cost-down SNES). Eventually it might be also possible to use composite video by connecting a matching oscillator directly to the S-RGB A chip (NTSC:3.579545MHz, PAL:4.43361875MHz) (not tested).

SNES Controller Mods

Shift Registers

SNES Joypads are basically consisting of buttons wired to a 16bit shift register. This could be reproduced using two 4021 chips (two 8bit parallel-in serial-out shift registers).

SNESPAD (SNES Controller to PC Parallel Port)

This is a circuit for connecting up to five SNES joypads to a PC Parallel Port, using 25pin DSUB or 36pin Centronics connector. The circuit can be used with drivers like "Direct Pad Pro" or "PPJoy", or by emulators with built-in SNESPAD support.

```
  Pin DB25  CNTR
  d3  5     5 ---|>|--.            .---.
  d4  6     6 ---|>|--+------------| O | 1 vcc
  d5  7     7 ---|>|--|  .---------| O | 2 clk
  d6  8     8 ---|>|--|  | .-------| O | 3 stb
  d7  9     9 ---|>|--'  | | .-----|_O_| 4 dta1
  d0  2     2 -----------' | |     | O | 5 dta3
  d1  3     3 -------------' |     | O | 6 io
  x   x     x ---------------' .---| O | 7 gnd
  gnd 18-25 19-30 -------------'    \_/
```

For Pad 1..5, wire Pin "x" to ack,pe,slct,err,busy (aka DB25 pin 10, 12, 13, 15, 11) (aka CNTR pin 10, 12, 13, 32, 11) (aka bit6, bit5, bit4, bit3, NOT(bit7) in the PC's I/O Port). The circuit is pretty well standarized (there is only one variant, a so-called "Linux" circuit with messed-up pin ordering: ack,busy,pe,slct,err for pad 1..5).

7pin Connectors (1mm pin diameter)

With some efforts, these can be made pulling contacts from regular DSUB connectors (which have same pin diameter, but different pin spacing). Solder the contacts onto a piece of board, and eventually build some plastic block with holes/notches as in real SNES connectors. Alternately, SNES extension cables (with one male & one female connector) are reportedly available.

SNES Xboo Upload (WRAM Boot)

```
                                  ____
CTR.01./STB-----|AND \_____WRAM.58./PAWR    VCC-------/cut/--CPU.81.RDY
EXT.09./PAWR----|____/                     PA7-------/cut/--WRAM.50.PA7
CTR.36./SELECT--|XOR \_____HIGHSEL         /PAWR-----/cut/--WRAM.58./PAWR
EXT.20.VCC -----|____/                     /WRAMSEL--/cut/--WRAM.15./WRAMSEL
HIGHSEL---------|OR  \_____WRAM.50.PA7     /ROMSEL---/cut/--SLT.49./ROMSEL
EXT.08.PA7 -----|____/                     CTR.01./STB-----[10K]--EXT.20.VCC
HIGHSEL---------|OR  \___ ____             CTR.14./AUTOLF--[10K]--EXT.20.VCC
SLT.32./WRAMSEL-|____/   |AND \___WRAM.15. CTR.36./SELECT--[10K]--EXT.20.VCC
CPU.77./ROMSEL--|OR  \___|____/   /WRAMSEL CTR.01./STB-----|10n|--EXT.23.GND
CTR.14./AUTOLF--|____/  ________*          CTR.14./AUTOLF--|10n|--EXT.23.GND
CTR.14./AUTOLF--|XOR \_|_ ____             CTR.36./SELECT--|10n|--EXT.23.GND
EXT.20.VCC------|____/   |OR  \___SLT.49.   CTR.31.INIT----------RESET_BUTTON
CPU.77./ROMSEL__________|____/   /ROMSEL   CTR.36./SELECT---------CPU.81.RDY
CTR.19-30.GND_________________EXT.23.GND   CTR.02-09.D0-D7---EXT.11-18.D0-D7
```

Extended Circuit (for larger ROMs, and for non-WRAM-boot-compatible ROMs)

```
                                  ____
CTR.14./AUTOLF--|AND \____(XOR)          /STB------/cut/-----------(AND)
CTR.01./STB-----|____/                    /AUTOLF---/cut/-----------(XOR)
CTR.01./STB-----|OR  \____(AND)          /PARD-----/cut/---WRAM.56./PARD
CTR.36./SELECT--|____/                    ________________
CPU.77./ROMSEL--|OR  \___________________|/CS   SRAM   CS2|__VCC (if any)
CTR.01./STB-----|____/        A0..A14___|A0..A14   D0..D7|__D0..D7
SLT.nn.A15------|XOR \___________________|A15          /OE|__CPU.92./RD
SLT.nn.A(hi+1)--|____/     A16..A(hi)___|A16..A(hi)   /WE|__CPU.91./WR
CTR.14./AUTOLF--|OR  \___ ____          |________________|
CTR.36./SELECT--|____/   |AND \___WRAM.56./PARD
EXT.10./PARD____________|____/
CTR.10./ACK_____________________CPU.39.OUT2
```

Revision Nov 2012 (/RD disable, for /ROMSEL-less carts like GSU)

```
                                  ____
* ------------|OR  \_____ SLT.23.     CPU.92./RD----/cut/---SLT.23./RD
CPU.92./RD ---|____/      /RD
```

Functional Description (WRAM Boot)
RESET/INIT and RDY/SELECT are used to reset and stop the CPU, in that state, PA0-PA7 and /WRAMSEL are all LOW (of which, PA7 and /WRAMSEL need to be changed for DMA), and WRAM address is reset to zero. Data is then DMA transferred to WRAM via databus and /STB/PAWR. Finally, WRAM is mirrored (in HiROM fashion) to the ROM-region via /AUTOLF, and entrypoint and program code are fetched from WRAM.
WRAM-Boot compatible files are identified by ID "XBOO" at FFE0h in ROM Header. WRAM-boot files should be also bootable from normal ROM (unless written by less-than-lame people).
The WRAM upload function is found in no$sns "Utility" menu. WRAM boot works only if a regular cartride is inserted, or if the lockout chip is disabled. Normal ROM-cartridges can be used if the cable is disconnected, or if the parallel port is set to /STB=HIGH, /AUTOLF=HIGH, /SELECT=HIGH, /INIT=LOW, and DATA=HIGH-Z.

Functional Description (Extended Circuit)
Data is uploaded (in 127.5K blocks) to WRAM, and is then automatically relocated (by a 0.5K stub) from WRAM to SRAM by software, /ACK is used to indicate completion of relocation, after uploading all block(s) the console gets reset with SRAM mapped to the ROM-region.
The XOR gate maps the bottom halves of the HiROM banks as additional LoROM banks. About A(hi) and A(hi+1): For example, for 128Kx8 chip (A0..A16): connect A(hi) to A16, A(hi+1) to A17. For more than 2Mx8 SRAM: connect the A(hi+1) input to GND; or leave out the XOR gate completely.
/STB and /AUTOLF are used (while /SELECT=HIGH) to select ROM, WRAM, or SRAM mapping, and as /PARD and /PAWR (while /SELECT=LOW). The /PARD pin allows to download status information and other data from WRAM.

Parts List

```
1  74LS08 Quad 2-Input AND gates
1  74LS32 Quad 2-Input OR gates
1  74LS86 Quad 2-Input XOR gates
3  10K Ohm Resistors (required when cable is disconnected)
3  10nF capacitor (to eliminate dirt on some ports)
1  36pin centronics socket (plus standard printer cable)
```

Additional components for Extended Circuit

```
1  74LS32 Quad 2-Input OR gates
1  nn-pin DIP Nx8 Static RAM (SRAM)
1  40-pin DIP Socket for SRAM
```

Requires a bi-directional parallel port (very old printer ports aren't bi-directional, also some hyper-modern ports aren't bi-directional when configured to "ECP" mode, for that ports: use "EPP" mode).
Currently SRAMs of up to 512Kx8 (32pin) should be available for less than $5, also 2Mx8 chips (36pin) are manufactured, but I've no idea where to buy them, best use a 40pin DIP socket for future expansion.

TEST Mode
Optionally, one can replace the "XBOO" ID-string in cartridge header by "TEST", this will cause normal Cartridge ROM to be mapped (instead of mirroring WRAM to the ROM area). The advantage is that the uploaded program can examine cartridge memory (eg. for reverse-engineering purposes), the disadvantage is that it cannot set up it's own IRQ/NMI vectors.
The program should redirect the reset vector from Bank 00h to Bank 7Eh by executing a JMP 7Exxxxh immediately after reset, and then wait until WRAM is no longer mapped at 008xxxxh; thereafter, ROM is freely accessible (the switch from WRAM to ROM mapping occurs circa 100us after RESET).

CPU 65XX Microprocessor

Overview

Instruction Set

Other Info
CPU Assembler Directives/Syntax
CPU Glitches
CPU The 65XX Family

CPU Registers and Flags

The 65XX CPUs are equipped with not more than three general purpose registers (A, X, Y). However, the limited number of registers is parts of covered by comfortable memory operations, especially page 0 of memory (address 0000h-00FFh) may be used for relative fast and complicated operations, in so far one might say that the CPU has about 256 8bit 'registers' (or 128 16bit 'registers') in memory. For details see Memory Addressing chapter.

Registers

```
Bits  Name  Expl.
8/16  A     Accumulator
8/16  X     Index Register X
8/16  Y     Index Register Y
16    PC    Program Counter
8/16  S     Stack Pointer (see below)
8     P     Processor Status Register (see below)
16    D     Zeropage Offset      ;expands 8bit  [nn]   to 16bit [00:nn+D]
8     DB    Data Bank            ;expands 16bit [nnnn] to 24bit [DB:nnnn]
8     PB    Program Counter Bank ;expands 16bit PC     to 24bit PB:PC
```

Stack Pointer
The stack pointer is addressing 256 bytes in page 1 of memory, ie. values 00h-FFh will address memory at 0100h-01FFh. As for most other CPUs, the stack pointer is decrementing when storing data. However, in the 65XX world, it points to the first FREE byte on stack, so, when initializing stack to top set S=(1)FFh (rather than S=(2)00h).

Processor Status Register (Flags)

```
Bit  Name  Expl.
0    C     Carry         (0=No Carry, 1=Carry)
1    Z     Zero          (0=Nonzero, 1=Zero)
2    I     IRQ Disable   (0=IRQ Enable, 1=IRQ Disable)
3    D     Decimal Mode  (0=Normal, 1=BCD Mode for ADC/SBC opcodes)
4    X/B   Break Flag    (0=IRQ/NMI, 1=BRK/PHP opcode)  (0=16bit, 1=8bit)
5    M/U   Unused        (Always 1)                     (0=16bit, 1=8bit)
6    V     Overflow      (0=No Overflow, 1=Overflow)
7    N     Negative/Sign (0=Positive, 1=Negative)
-    E                                                  (0=16bit, 1=8bit)
```

Emulation Flag (E) (can accessed only via XCE opcode)
Activates 6502 emulation mode. In that mode, A/X/Y are 8bit (as when X/M are set). S is 8bit (range 0100h..01FFh). The X/M flags are replaced by B/U flags (B=Break/see below, U=Unused/always 1). Exception Vectors are changed to 6502 addresses, and exceptions push only 16bit return addresses (rather than 24bit). Conditional Branches take one extra cycle when crossing pages.
The new 65C02 and 65C816 opcodes can be kept used; respectively, none of the undocumented opcodes of the original 6502 CPU are supported.

Accumulator/Memory Flag (M)
Switches A to 8bit mode (the upper 8bit of A can be still accessed via XBA).
Additionally switches all opcodes that do not use A,X,Y operands (eg. STZ, and read-modify-write opcodes like inc/dec, Rotate/Shift) to 8bit mode.

Index Register Flag (X)
Switches X and Y to 8bit mode (the upper 8bit are forcefully set to 00h).

Carry Flag (C)
Caution: When used for subtractions (SBC and CMP), the carry flag is having opposite meaning as for normal 80x86 and Z80 CPUs, ie. it is SET when above-or-equal. For all other instructions (ADC, ASL, LSR, ROL, ROR) it works as normal, whereas ROL/ROR are rotating <through> carry (ie. much like 80x86 RCL/RCR and not like ROL/ROR).

Zero Flag (Z), Negative/Sign Flag (N), Overflow Flag (V)
Works just as everywhere, Z it is set when result (or destination register, in case of some 'move' instructions) is zero, N is set when signed (ie. same as Bit 7 of result/destination). V is set when an addition/subtraction exceeded the maximum range for signed numbers (-128..+127).

IRQ Disable Flag (I)
Disables IRQs when set. NMIs (non maskable interrupts) and BRK instructions cannot be disabled.

Decimal Mode Flag (D)
Packed BCD mode (range 00h..99h) for ADC and SBC opcodes.

Break Flag (B)
The Break flag is intended to separate between IRQ and BRK which are both using the same vector, [FFFEh]. The flag cannot be accessed directly, but there are 4 situations which are writing the P register to stack, which are then allowing the examine the B-bit in the pushed value: The BRK and PHP opcodes always write "1" into the bit, IRQ/NMI execution always write "0".

Common Confusing Aliases for Register Names

```
DPR   D  (used in homebrew specs)
K     PB (used by PHK)
PBR   PB (used in specs)
DBR   DB (used in specs)
B     DB (used by PLB,PHB)
B     Upper 8bit of A (used by XBA)
C     Full 16bit of A (used by TDC,TCD,TSC,TCS)
```

CPU Memory Addressing

```
Name           Native   Nocash
Implied        -        A,X,Y,S,P
Immediate      #nn      nn
Zero Page      nn       [nn]
Zero Page,X    nn,X     [nn+X]
Zero Page,Y    nn,Y     [nn+Y]
Absolute       nnnn     [nnnn]
Absolute,X     nnnn,X   [nnnn+X]
Absolute,Y     nnnn,Y   [nnnn+Y]
(Indirect,X)   (nn,X)   [[nn+X]]
(Indirect),Y   (nn),Y   [[nn]+Y]
```

Zero Page - [nn] [nn+X] [nn+Y]
Uses an 8bit parameter (one byte) to address the first 256 of memory at 0000h..00FFh. This limited range is used even for "nn+X" and "nn+Y", ie. "C0h+60h" will access 0020h (not 0120h).

Absolute - [nnnn] [nnnn+X] [nnnn+Y]
Uses a 16bit parameter (two bytes) to address the whole 64K of memory at 0000h..FFFFh. Because of the additional parameter bytes, this is a bit slower than Zero Page accesses.

Indirect - [[nn+X]] [[nn]+Y]
Uses an 8bit parameter that points to a 16bit parameter in page zero.
Even though the CPU doesn't support 16bit registers (except for the program counter), this (double-)indirect addressing mode allows to use variable 16bit pointers.

On-Chip Bi-directional I/O port
Addresses (00)00h and (00)01h are occupied by an I/O port which is built-in into 6510, 8500, 7501, 8501 CPUs (eg. used in C64 and C16), be sure not to use the addresses as normal memory. For description read chapter about I/O ports.

Caution
Because of the identical format, assemblers will be more or less unable to separate between [XXh+r] and [00XXh+r], the assembler will most likely produce [XXh+r] when address is already known to be located in page 0, and [00XXh+r] in case of forward references.
Beside for different opcode size/time, [XXh+r] will always access page 0 memory (even when XXh+r>FFh), while [00XXh+r] may direct to memory in page 0 or 1, to avoid unpredictable results be sure not to use (00)XXh+r>FFh if possible.

CPU Clock Cycles

SNES Memory Speed

```
Memory Area      Speed          Clks Comment
00-3F:0000-1FFF  Medium 2.68MHz 8    WRAM (8K mirror of 7E:0000-1FFF)
00-3F:2000-3FFF  Fast   3.58MHz 6    I/O and Expansion
00-3F:4000-41FF  Slow   1.78MHz 12   Manual Joypad Reading
00-3F:4200-5FFF  Fast   3.58MHz 6    I/O and Expansion
00-3F:6000-7FFF  Medium 2.68MHz 8    SRAM and Expansion
00-3F:8000-FFFF  Medium 2.68MHz 8    ROM (32K banks)
40-7F:0000-FFFF  Medium 2.68MHz 8    ROM/SRAM/WRAM
80-BF:0000-1FFF  Medium 2.68MHz 8    WRAM (8K mirror of 7E:0000-1FFF)
80-BF:2000-3FFF  Fast   3.58MHz 6    I/O and Expansion
80-BF:4000-41FF  Slow   1.78MHz 12   Manual Joypad Reading
80-BF:4200-5FFF  Fast   3.58MHz 6    I/O and Expansion
80-BF:6000-7FFF  Medium 2.68MHz 8    SRAM and Expansion
80-BF:8000-FFFF  Variable       6/8  ROM (32K banks)  ;\speed selectable
C0-FF:0000-FFFF  Variable       6/8  ROM (64K banks)  ;/via port 420Dh
Internal Cycles  Fast   3.58MHz 6    Internal cycles (eg. 2nd cycle in NOPs)
```

Implied or Immediate Operands

```
CN        2     Opcodes without memory/immediate parameters
CNN       3     XBA/WAI/STP (swap A, wait irq, stop)
CPp       2,3   nn or nnnn                                  ;+p if 16bit
CPN       3     nn  (REP/SEP)
```

Memory Operands

```
CPnDd     3,4,5 [nn+d]                ;+n if (D AND 00FFh)>0, +d  if 16bit
CPnDdNDd  5..8  [nn+d] (RMW)          ;+n if (D AND 00FFh)>0, +dd if 16bit
CPnNDd    4,5,6 [nn+x+d] or [nn+y+d]  ;+n if (D AND 00FFh)>0, +d  if 16bit
CPnNDdNDd 6..9  [nn+x+d] (RMW)        ;+n if (D AND 00FFh)>0, +dd if 16bit
CPNDd     4,5   [nn+s]                                      ;+d  if 16bit
CPPDd     4,5   [nnnn]                                      ;+d  if 16bit
CPPDdNDd  6,8   [nnnn] (RMW)                                ;+dd if 16bit
CPPyDd    4,5,6 [nnnn+x] or [nnnn+y] (RD)    +y     xxx     ;+d  if 16bit
CPPNDd    5,6   [nnnn+x] or [nnnn+y] (WR)                   ;+d  if 16bit
CPPNDdNDd 7,9   [nnnn+x] (RMW)                              ;+dd if 16bit
CPPPDd    5,6   [nnnnnn] or [nnnnnn+x]                      ;+d  if 16bit
CPnAADd   5,6,7 [[nn+d]]              ;+n if (D AND 00FFh)>0, +d  if 16bit
CPnAAyDd  5..8  [[nn+d]+y] (RD)             +n+y   xxx      ;+d  if 16bit
CPnAANDd  6..8  [[nn+d]+y] (WR)       ;+n if (D AND 00FFh)>0, +d  if 16bit
CPnNAADd  6,7,8 [[nn+d+x]]            ;+n if (D AND 00FFh)>0, +d  if 16bit
CPNAANDd  7,8   [[nn+s]+y]                                  ;+d  if 16bit
CPnAAADd  6,7,8 [far[nn+d]] or [far[nn+d]+y]  ;+n if (D ..)>0, +d  if 16bit
CPPDDNN   7     ldir/lddr
```

Push/Pop

```
CNsS      3,4   PUSH register                               ;+s  if 16bit
CNNSs     4,5   POP register                                ;+s  if 16bit
CPnDDSS   6,7   PUSH word[nn+d] ("PEI") ;+n if (D AND 00FFh)>0
CPPSS     5     PUSH nnnn       ("PEA")
CPPNSS    6     PUSH $+/-nnnn   ("PER")
```

Jumps

```
CP        2     relative 8bit jump (condition false)
CPNx      3,4   relative 8bit jump   ;+x if "E=1 and crossing 100h-boundary"
CPPN      4     relative 16bit jump
CPP       3     Jump nnnn
```

```
CPPNSS    6     Call nnnn
CPPSNPSS  8(7?) Call nnnnnn
CPPDD     5     jump [nnnn]
CPPNDD    6     jump [nnnn+X]
CPPDDD    6     jump far[nnnn]
CPSSPNDD  8     call [nnnn+X]
CNNSSN    6     RTS (ret)
CNNSSS    6     RTL (retf)
CNNSSSs   6,7   RTI (reti)                              ;+s if E=0
NNsSSSDD  7,8   Exception (/ABORT, /IRQ, /NMI, /RES)    ;+s if E=0
CPsSSSDD  7,8   Exception (BRK, COP)                    ;+s if E=0
```

Legend

```
C   Opcode command
P   Opcode parameter (immediate or address)
A   Address cycles (on double-indirect addresses)
D   Data cycles
N   Internal cycles
S   Stack cycles
```

Additional cycles

```
d   Data (MSB in 16bit modes, ie. when M/X=0)
s   Stack (MSB in 16bit modes, or BANK in 65C816-mode exceptions)
n   Internal cycle, when (D AND 00FFh)>0
x   Internal cycle, when E=1 and rel-jump crossing 100h-boundary
y   Internal cycle, when X=0 or indexing across page boundaries
```

CPU Memory and Register Transfers

Register to Register Transfer

```
Opcode    Flags Clks  Native  Nocash    Bits  Effect
A8        nz----  2   TAY     MOV Y,A   x     Y=A
AA        nz----  2   TAX     MOV X,A   x     X=A
BA        nz----  2   TSX     MOV X,S   x     X=S
98        nz----  2   TYA     MOV A,Y   m     A=Y
8A        nz----  2   TXA     MOV A,X   m     A=X
9A        ------  2   TXS     MOV S,X   e     S=X
9B        nz----  2   TXY     MOV Y,X   x     Y=X
BB        nz----  2   TYX     MOV X,Y   x     X=Y
7B        nz----  2   TDC     MOV A,D   16    A=D
5B        nz----  2   TCD     MOV D,A   16    D=A
3B        nz----  2   TSC     MOV A,SP  16    A=SP
1B        ------  2   TCS     MOV SP,A  e?    SP=A
```

Load Register from Memory

```
Opcode      Flags Clks  Native       Nocash         Bits  Effect
A9 nn       nz----  2   LDA #nn      MOV A,nn          A=nn
A5 nn       nz----  3   LDA nn       MOV A,[nn]        A=[D+nn]
B5 nn       nz----  4   LDA nn,X     MOV A,[nn+X]      A=[D+nn+X]
A3 nn       nz----      LDA nn,S     MOV A,[nn+S]      A=[nn+S]
AD nn nn    nz----  4   LDA nnnn     MOV A,[nnnn]      A=[DB:nnnn]
BD nn nn    nz----  4*  LDA nnnn,X   MOV A,[nnnn+X]    A=[DB:nnnn+X]
B9 nn nn    nz----  4*  LDA nnnn,Y   MOV A,[nnnn+Y]    A=[DB:nnnn+Y]
AF nn nn nn nz----      LDA nnnnnn   MOV A,[nnnnnn]    A=[nnnnnn]
BF nn nn nn nz----      LDA nnnnnn,X MOV A,[nnnnnn+X]  A=[nnnnnn+X]
B2 nn       nz----      LDA (nn)     MOV A,[[nn]]      A=[WORD[D+nn]]
A1 nn       nz----  6   LDA (nn,X)   MOV A,[[nn+X]]    A=[WORD[D+nn+X]]
B1 nn       nz----  5*  LDA (nn),Y   MOV A,[[nn]+Y]    A=[WORD[D+nn]+Y]
B3 nn       nz----      LDA (nn,S),Y MOV A,[[nn+S]+Y]  A=[WORD[nn+S]+Y]
A7 nn       nz----      LDA [nn]     MOV A,[FAR[nn]]   A=[FAR[D+nn]]
B7 nn       nz----      LDA [nn],y   MOV A,[FAR[nn]+Y] A=[FAR[D+nn]+Y]
A2 nn       nz----  2   LDX #nn      MOV X,nn          X=nn
A6 nn       nz----  3   LDX nn       MOV X,[nn]        X=[D+nn]
B6 nn       nz----  4   LDX nn,Y     MOV X,[nn+Y]      X=[D+nn+Y]
AE nn nn    nz----  4   LDX nnnn     MOV X,[nnnn]      X=[DB:nnnn]
BE nn nn    nz----  4*  LDX nnnn,Y   MOV X,[nnnn+Y]    X=[DB:nnnn+Y]
A0 nn       nz----  2   LDY #nn      MOV Y,nn          Y=nn
A4 nn       nz----  3   LDY nn       MOV Y,[nn]        Y=[D+nn]
B4 nn       nz----  4   LDY nn,X     MOV Y,[nn+X]      Y=[D+nn+X]
AC nn nn    nz----  4   LDY nnnn     MOV Y,[nnnn]      Y=[DB:nnnn]
BC nn nn    nz----  4*  LDY nnnn,X   MOV Y,[nnnn+X]    Y=[DB:nnnn+X]
```

* Add one cycle if indexing crosses a page boundary.

Store Register in Memory

```
Opcode      Flags Clks  Native       Nocash         Bits  Effect
64 nn       ------  3   STZ nn       MOV [nn],0        m  [D+nn]=0
74 nn       ------  4   STZ nn_x     MOV [nn+X],0      m  [D+nn+X]=0
9C nn nn    ------  4   STZ nnnn     MOV [nnnn],0      m  [DB:nnnn]=0
9E nn nn    ------  5   STZ nnnn_x   MOV [nnnn+X],0    m  [DB:nnnn+X]=0
85 nn       ------  3   STA nn       MOV [nn],A        m  [D+nn]=A
95 nn       ------  4   STA nn,X     MOV [nn+X],A      m  [D+nn+X]=A
83 nn       ------      STA nn,S     MOV [nn+S],A      m  [nn+S]=A
8D nn nn    ------  4   STA nnnn     MOV [nnnn],A      m  [DB:nnnn]=A
9D nn nn    ------  5   STA nnnn,X   MOV [nnnn+X],A    m  [DB:nnnn+X]=A
99 nn nn    ------  5   STA nnnn,Y   MOV [nnnn+Y],A    m  [DB:nnnn+Y]=A
8F nn nn nn ------      STA nnnnnn   MOV [nnnnnn],A    m  [nnnnnn]=A
9F nn nn nn ------      STA nnnnnn,X MOV [nnnnnn+X],A  m  [nnnnnn+X]=A
81 nn       ------  6   STA (nn,X)   MOV [[nn+X]],A    m  [WORD[D+nn+X]]=A
91 nn       ------  6   STA (nn),Y   MOV [[nn]+Y],A    m  [WORD[D+nn]+Y]=A
92 nn       ------      STA (nn)     MOV [[nn]],A      m  [WORD[D+nn]]=A
93 nn       ------      STA (nn,S),Y MOV [[nn+S]+Y],A  m  [WORD[nn+S]+Y]=A
87 nn       ------      STA [nn]     MOV [FAR[nn]],A   m  [FAR[D+nn]]=A
97 nn       ------      STA [nn],y   MOV [FAR[nn]+Y],A m  [FAR[D+nn]+Y]=A
86 nn       ------  3   STX nn       MOV [nn],X        x  [D+nn]=X
96 nn       ------  4   STX nn,Y     MOV [nn+Y],X      x  [D+nn+Y]=X
8E nn nn    ------  4   STX nnnn     MOV [nnnn],X      x  [DB:nnnn]=X
84 nn       ------  3   STY nn       MOV [nn],Y        x  [D+nn]=Y
94 nn       ------  4   STY nn,X     MOV [nn+X],Y      x  [D+nn+X]=Y
```

Push/Pull (Stack)

Opcode	Flags	Clks	Native	Nocash	Bits	Effect
48	------	3	PHA	PUSH A	m	[S]=A
DA	------	3	PHX	PUSH X	x	[S]=X
5A	------	3	PHY	PUSH Y	x	[S]=Y
08	------	3	PHP	PUSH P	8	[S]=P
8B	------	3	PHB	PUSH DB	8	[S]=DB
4B	------	3	PHK	PUSH PB	8	[S]=PB
0B	------	4	PHD	PUSH D	16	[S]=D
D4 nn	------	6	PEI nn	PUSH WORD[nn]	16	[S]=WORD[D+nn]
F4 nn nn	------	5	PEA nnnn	PUSH nnnn	16	[S]=NNNN
62 nn nn	------	6	PER rel16	PUSH disp16	16	[S]=$+3+disp
68	nz----	4	PLA	POP A	m	A=[S]
FA	nz----	4	PLX	POP X	x	X=[S]
7A	nz----	4	PLY	POP Y	x	Y=[S]
2B	nz----	5	PLD	POP D	16	D=[S]
AB	nz----	4	PLB	POP DB	8	DB=[S]
28	nzcidv	4	PLP	POP P	8	P=[S]

Notes: PLA sets Z and N according to content of A. The B-flag and unused flags cannot be changed by PLP, these flags are always written as "1" by PHP.

Memory Block Transfer Commands

Opcode	Flags	Clks	Native	Nocash	;Notes
44 dd ss	------	7x	MVP ss,dd	LDDR [dd:Y],[ss:X],A+1	;DEC X/Y
54 dd ss	------	7x	MVN ss,dd	LDIR [dd:Y],[ss:X],A+1	;INC X/Y

Sets DB=dstbank (!), copies one byte from [srcbank:X] to [dstbank:Y], increments or decrements X and Y (8bit or 16bit, depending on X and E flags), decrements A (all 16bits, no matter of M and E flags), continues at next opcode when resulting A=FFFFh, otherwise sets PC=PC-3 (thus repeats the transfer opcode).
Note: The native opcode names (MVP/MVN, Move Positive/Negative) are chosen so that "positive" refers to transfers from "to higher" memory addresses (which would matter on overlapping source/dest blocks), accordingly, the practical meaning is that "positive" means to DECREASE the X,Y registers.

CPU Arithmetic/Logical Operations

ALU Opcodes

Base	Flags	Native	Nocash	Operands	Name	Function
00	nz----	ORA op	OR A,op	<alu_types>	OR	A=A OR op
20	nz----	AND op	AND A,op	<alu_types>	AND	A=A AND op
40	nz----	EOR op	XOR A,op	<alu_types>	XOR	A=A XOR op
60	nzc--v	ADC op	ADC A,op	<alu_types>	Add	A=A+C+op
E0	nzc--v	SBC op	SBC A,op	<alu_types>	Subtract	A=A+C-1-op
C0	nzc---	CMP op	CMP A,op	<alu_types>	Compare	A-op
E0	nzc---	CPX op	CMP X,op	<cpx_types>	Compare	X-op
C0	nzc---	CPY op	CMP Y,op	<cpx_types>	Compare	Y-op

alu_types (Operands for OR,AND,XOR,ADC,SBC,CMP Opcodes)

Opcode	Clks	Native	Nocash	Name	Effect
Base+09 nn	2	#nn	nn	Immediate	nn
Base+05 nn	3	nn	[nn]	Zero Page	[D+nn]
Base+15 nn	4	nn,X	[nn+X]	Zero Page,X	[D+nn+X]
Base+0D nn nn	4	nnnn	[nnnn]	Absolute	[DB:nnnn]
Base+1D nn nn	4*	nnnn,X	[nnnn+X]	Absolute,X	[DB:nnnn+X]
Base+19 nn nn	4*	nnnn,Y	[nnnn+Y]	Absolute,Y	[DB:nnnn+Y]
Base+01 nn	6	(nn,X)	[[nn+X]]	(Indirect,X)	[WORD[D+nn+X]]
Base+11 nn	5*	(nn),Y	[[nn]+Y]	(Indirect),Y	[WORD[D+nn]+Y]
Base+12 nn		(nn)	[[nn]]	(Indirect)	[WORD[D+nn]]
Base+03 nn		nn,S	[nn+S]		[nn+S]
Base+13 nn		(nn,S),Y	[[nn+S]+Y]		[WORD[nn+S]+Y]
Base+07 nn		[nn]	[FAR[nn]]		[FAR[D+nn]]
Base+17 nn		[nn],y	[FAR[nn]+Y]		[FAR[D+nn]+Y]
Base+0F nn nn nn		nnnnnn	[nnnnnn]		[nnnnnn]
Base+1F nn nn nn		nnnnnn,X	[nnnnnn+X]		[nnnnnn+X]

* Add one cycle if indexing crosses a page boundary.

cpx_types (Operands for CMP Opcodes with X,Y Operand)

Opcode	Clks	Native	Nocash	Name	Effect
Base+00 nn	2	#nn	nn	Immediate	nn
Base+04 nn	3	nn	[nn]	Zero Page	[D+nn]
Base+0C nn nn	4	nnnn	[nnnn]	Absolute	[DB:nnnn]

Bit Test

Opcode	Flags	Clks	Native	Nocash	Operand
24 nn	xz---x	3	BIT nn	TEST A,[nn]	[D+nn]
2C nn nn	xz---x	4	BIT nnnn	TEST A,[nnnn]	[DB:nnnn]
34 nn	xz---x		BIT nn,X	TEST A,[nn+X]	[D+nn+X]
3C nn nn	xz---x		BIT nnnn,X	TEST A,[nnnn+X]	[DB:nnnn+X]
89 nn	-z----		BIT #nn	TEST A,nn	nn

Flags are set as "Z=((A AND op)=0)", and "N=op.Bit(MSB)", and "V=op.Bit(MSB-1)". Where MSB=bit15/bit7, and MSB-1=bit14/bit6 (depending on M-flag).
Note that N and V do rely only on "op" (ie. not on "A AND op").

Increment by one

Opcode	Clks	Native	Nocash	Effect
E6 nn	nz----	5	INC nn	INC [nn]	[D+nn]=[D+nn]+1
F6 nn	nz----	6	INC nn,X	INC [nn+X]	[D+nn+X]=[D+nn+X]+1
EE nn nn	nz----	6	INC nnnn	INC [nnnn]	[DB:nnnn]=[DB:nnnn]+1
FE nn nn	nz----	7	INC nnnn,X	INC [nnnn+X]	[DB:nnnn+X]=[DB:nnnn+X]+1
E8	nz----	2	INX	INC X	X=X+1
C8	nz----	2	INY	INC Y	Y=Y+1
1A	nz----	2	INA	INC A	A=A+1

Decrement by one

Opcode	Clks	Native	Nocash	Effect
C6 nn	nz----	5	DEC nn	DEC [nn]	[D+nn]=[D+nn]-1
D6 nn	nz----	6	DEC nn,X	DEC [nn+X]	[D+nn+X]=[D+nn+X]-1
CE nn nn	nz----	6	DEC nnnn	DEC [nnnn]	[DB:nnnn]=[DB:nnnn]-1
DE nn nn	nz----	7	DEC nnnn,X	DEC [nnnn+X]	[DB:nnnn+X]=[DB:nnnn+X]-1
CA	nz----	2	DEX	DEC X	X=X-1

```
  88        nz----  2   DEY        DEC Y        Y=Y-1
  3A        nz----  2   DEA        DEC A        A=A-1
```

TSB/TRB (Test and Set/Reset)

```
  Opcode          Clks  Native      Nocash
  04 nn     -z----  5   TSB nn      SET [nn],A    ;\"TEST op,A" --> z
  0C nn nn  -z----  6   TSB nnnn    SET [nnnn],A  ;/then "OR op,A"
  14 nn     -z----  5   TRB nn      CLR [nn],A    ;\"TEST op,A" --> z
  1C nn nn  -z----  6   TRB nnnn    CLR [nnnn],A  ;/then "AND op,NOT A"
```

CPU Rotate and Shift Instructions

Shift Left Logical/Arithmetic

```
  Opcode          Clks  Native      Nocash       Effect
  0A        nzc---  2   ASL A       SHL A        SHL A
  06 nn     nzc---  5   ASL nn      SHL [nn]     SHL [D+nn]
  16 nn     nzc---  6   ASL nn,X    SHL [nn+X]   SHL [D+nn+X]
  0E nn nn  nzc---  6   ASL nnnn    SHL [nnnn]   SHL [DB:nnnn]
  1E nn nn  nzc---  7   ASL nnnn,X  SHL [nnnn+X]  SHL [DB:nnnn+X]
```

Shift Right Logical

```
  4A        0zc---  2   LSR A       SHR A        SHR A
  46 nn     0zc---  5   LSR nn      SHR [nn]     SHR [D+nn]
  56 nn     0zc---  6   LSR nn,X    SHR [nn+X]   SHR [D+nn+X]
  4E nn nn  0zc---  6   LSR nnnn    SHR [nnnn]   SHR [DB:nnnn]
  5E nn nn  0zc---  7   LSR nnnn,X  SHR [nnnn+X]  SHR [DB:nnnn+X]
```

Rotate Left through Carry

```
  2A        nzc---  2   ROL A       RCL A        RCL A
  26 nn     nzc---  5   ROL nn      RCL [nn]     RCL [D+nn]
  36 nn     nzc---  6   ROL nn,X    RCL [nn+X]   RCL [D+nn+X]
  2E nn nn  nzc---  6   ROL nnnn    RCL [nnnn]   RCL [DB:nnnn]
  3E nn nn  nzc---  7   ROL nnnn,X  RCL [nnnn+X]  RCL [DB:nnnn+X]
```

Rotate Right through Carry

```
  6A        nzc---  2   ROR A       RCR A        RCR A
  66 nn     nzc---  5   ROR nn      RCR [nn]     RCR [D+nn]
  76 nn     nzc---  6   ROR nn,X    RCR [nn+X]   RCR [D+nn+X]
  6E nn nn  nzc---  6   ROR nnnn    RCR [nnnn]   RCR [DB:nnnn]
  7E nn nn  nzc---  7   ROR nnnn,X  RCR [nnnn+X]  RCR [DB:nnnn+X]
```

Notes:
ROR instruction is available on MCS650X microprocessors after June, 1976.
ROL and ROR rotate an 8bit value through carry (rotates 9bits in total).

CPU Jump and Control Instructions

Normal Jumps

```
  Opcode       Flags Clks  Native       Nocash        Effect
  80 dd        ------  3xx BRA disp8    JMP disp      PC=PC+/-disp8
  82 dd dd     ------  4   BRL disp16   JMP disp      PC=PC+/-disp16
  4C nn nn     ------  3   JMP nnnn     JMP nnnn      PC=nnnn
  5C nn nn nn  ------  4   JMP nnnnnn   JMP nnnnnn    PB:PC=nnnnnn
  6C nn nn     ------  5   JMP (nnnn)   JMP [nnnn]    PC=WORD[00:nnnn]
  7C nn nn     ------  6   JMP (nnnn,X) JMP [nnnn+X]  PC=WORD[PB:nnnn+X]
  DC nn nn     ------  6   JML ...      JMP FAR[nnnn] PB:PC=[00:nnnn]
  20 nn nn     ------  6   JSR nnnn     CALL nnnn     [S]=PC+2,PC=nnnn
  22 nn nn nn  ------  4   JSL nnnnnn   CALL nnnnnn   PB:PC=nnnnnn [S]=PB:PC+3
  FC nn nn     ------  6   JSR (nnnn,X) CALL [nnnn+X] PC=WORD[PB:nnnn+X] [S]=PC
  40           nzcidv  6   RTI          RETI          P=[S+1],PB:PC=[S+2],S=S+4
  6B           ------  ?   RTL          RETF          PB:PC=[S+1]+1, S=S+3
  60           ------  6   RTS          RET           PC=[S+1]+1, S=S+2
```

Note: RTI cannot modify the B-Flag or the unused flag.
Glitch: For JMP [nnnn] the operand word cannot cross page boundaries, ie. JMP [03FFh] would fetch the MSB from [0300h] instead of [0400h]. Very simple workaround would be to place a ALIGN 2 before the data word.

Conditional Branches (Branch on condition, to PC=PC+/-nn)

```
  Opcode    Flags Clks  Native  Nocash        Condition (jump if)
  10 dd     ------  2** BPL     JNS     disp  ;N=0 (plus/positive)
  30 dd     ------  2** BMI     JS      disp  ;N=1 (minus/negative/signed)
  50 dd     ------  2** BVC     JNO     disp  ;V=0 (no overflow)
  70 dd     ------  2** BVS     JO      disp  ;V=1 (overflow)
  90 dd     ------  2** BCC/BLT JNC/JB  disp  ;C=0 (less/below/no carry)
  B0 dd     ------  2** BCS/BGE JC/JAE  disp  ;C=1 (above/greater/equal/carry)
  D0 dd     ------  2** BNE/BZC JNZ/JNE disp  ;Z=0 (not zero/not equal)
  F0 dd     ------  2** BEQ/BZS JZ/JE   disp  ;Z=1 (zero/equal)
```

** The execution time is 2 cycles if the condition is false (no branch executed). Otherwise, 3 cycles if the destination is in the same memory page, or 4 cycles if it crosses a page boundary (see below for exact info).
Note: After subtractions (SBC or CMP) carry=set indicates above-or-equal, unlike as for 80x86 and Z80 CPUs.

Interrupts, Exceptions, Breakpoints

```
  Opcode                                                  6502     65C816
  00   BRK    Break      B=1 [S]=$+2,[S]=P,D=0 I=1, PB=00, PC=[00FFFE] [00FFE6]
  02   COP    ;65C816    B=1 [S]=$+2,[S]=P,D=0 I=1, PB=00, PC=[00FFF4] [00FFE4]
  --   /ABORT ;65C816                               PB=00, PC=[00FFF8] [00FFE8]
  --   /IRQ   Interrupt  B=0 [S]=PC, [S]=P,D=0 I=1, PB=00, PC=[00FFFE] [00FFEE]
  --   /NMI   NMI        B=0 [S]=PC, [S]=P,D=0 I=1, PB=00, PC=[00FFFA] [00FFEA]
  --   /RESET Reset      D=0 E=1 I=1 D=0000, DB=00  PB=00, PC=[00FFFC] N/A
```

Notes: IRQs can be disabled by setting the I-flag, a BRK command, a NMI, and a /RESET signal cannot be masked by setting I.
Exceptions do first change the B-flag (in 6502 mode), then write P to stack, and then set the I-flag, the D-flag IS cleared (unlike as on original 6502).
In 6502 mode, the same vector is shared for BRK and IRQ, software can separate between BRK and IRQ by examining the pushed B-flag only.

BRK opcode.
Software or hardware must take care to acknowledge or reset /IRQ or /NMI signals after processing it.
```
IRQs are executed whenever "/IRQ=LOW AND I=0".
NMIs are executed whenever "/NMI changes from HIGH to LOW".
```
If /IRQ is kept LOW then same (old) interrupt is executed again as soon as setting I=0. If /NMI is kept LOW then no further NMIs can be executed.

CPU Control
```
Opcode    Flags Clks  Native    Nocash    Effect
18        --0---  2   CLC       CLC       C=0    ;Clear carry flag
58        ---0--  2   CLI       EI        I=0    ;Clear interrupt disable bit
D8        ----0-  2   CLD       CLD       D=0    ;Clear decimal mode
B8        -----0  2   CLV       CL?       V=0    ;Clear overflow flag
38        --1---  2   SEC       STC       C=1    ;Set carry flag
78        ---1--  2   SEI       DI        I=1    ;Set interrupt disable bit
F8        ----1-  2   SED       STD       D=1    ;Set decimal mode
C2 nn     xxxxxx  3   REP #nn   CLR P,nn  P=P AND NOT nn
E2 nn     xxxxxx  3   SEP #nn   SET P,nn  P=P OR nn
FB        --c---  2   XCE       XCE       C=E, E=C
```

Special Opcodes
```
Opcode    Flags Clks  Native      Nocash
DB        ------  -   STP         KILL      ;STOP/KILL
EB        nz----  3   XBA         SWAP A    ;A=B, B=A, NZ=LSB
CB        ------  3x  WAI         HALT      ;HALT
42 nn             2   WDM #nn     NUL nn    ;No operation
EA        ------  2   NOP         NOP       ;No operation
```
WAI/HALT stops the CPU until an exception (usually an IRQ or NMI) request occurs; in case of IRQs this works even if IRQs are disabled (via I=1).

Conditional Branch Page Crossing
The branch opcode with parameter takes up two bytes, causing the PC to get incremented twice (PC=PC+2), without any extra boundary cycle. The signed parameter is then added to the PC (PC+disp), the extra clock cycle occurs if the addition crosses a page boundary (next or previous 100h-page).

CPU Assembler Directives/Syntax

Below are some common 65XX assembler directives, and the corresponding expressions in 80XX-style language.

```
65XX-style    80XX-style         Expl.
.native       .nocash            select native or nocash syntax
*=$c100       org 0c100h         sets the assumed origin in memory
*=*+8         org $+8            increments origin, does NOT produce data
label         label:             sets a label equal to the current address
label=$dc00   label equ 0dc00h   assigns a value or address to label
.by $00       db 00h             defines a (list of) byte(s) in memory
.byt $00      defb 00h           same as .by and db
.wd $0000     dw 0000h           defines a (list of) word(s) in memory
.end          end                indicates end of source code file
|nn          [|nn]              force 16bit "00NN" instead 8bit "NN"
#<nnnn        nnnn AND 0FFh      isolate lower 8bits of 16bit value
#>nnnn        nnnn DIV 100h      isolate upper 8bits of 16bit value
```

Special Directives
```
.65xx       Select 6502 Instruction Set
.nes        Create NES ROM-Image with .NES extension
.c64_prg    Create C64 file with .PRG extension/stub/fixed entry
.c64_p00    Create C64 file with .P00 extension/stub/fixed entry/header
.vic20_prg  Create VIC20/C64 file with .PRG extension/stub/relocated entry
end entry   End of Source, the parameter specifies the entrypoint
```
The C64 files contain Basic Stub "10 SYS<entry>" with default ORG 80Eh.

VIC20 Stub
The VIC20 Stub is "10 SYSPEEK(44)*256+<entry>" with default ORG 1218h, this relocates the entryoint relative to the LOAD address (for C64: 818h, for VIC20: 1018h (Unexpanded), 0418h (3K Expanded), 1218h (8K and more Expansion). It does NOT relocate absolute addresses in the program, if the program wishes to run at a specific memory location, then it must de-relocate itself from the LOAD address to the desired address.

CPU Glitches

Dummy Write Cycles in Read-Modify-Opcodes
The original 6502 CPU includes a dummy-write cycle in all Read-Modify opcodes (ie. INC, DEC, Shift/Rotate). The 65C816 does NOT reproduce this effect (neither when E=0, nor when E=1).

Dummy Read Cycles at Page-Wraps
```
[Below applies to original 6502 CPUs]
[Unknown if 65C816 has similar effects on page- and/or bank-wraps?]
```
Dummy reads occur when reads from [nnnn+X] or [nnnn+Y] or [WORD[nn]+Y] are crossing page boundaries, this applies only to raw read-opcodes, not for write or read-and-modify opcodes (ie. only for opcodes that include an extra clock cycle on page boundaries, such as LDA, CMP, etc.)
For above reads, the CPU adds the index register to the lower 8bits of the 16bit memory address, and does then read from the resulting memory address, if the addition caused a carry-out, then an extra clock cycle is used to increment the upper 8bits of the address, and to read from the correct memory location. For example, a read from [1280h+X] with X=C0h produces a dummy read from [1240h], followed by the actual read from [1340h].
Dummy reads cause no problems with normal ROM or RAM, but may cause problems with read-sensitive I/O ports (eg. IRQ flags that are automatically cleared after reading, or data-registers that are automatically incrementing associated memory pointers, etc.)

CPU The 65XX Family

Different versions of the 6502:

All of these processors are the same concerning the software-side:
```
6501   Some sort of 6502 prototype
```

```
6507   Used in Atari 2600, 28pins (only 13 address lines, no /IRQ, no /NMI).
6510   Used in C64, with built-in 6bit I/O port.
7501   Used in C16,C116,Plus/4, with built-in 7bit I/O Port, without /NMI pin.
8500   Used in C64-II, with different pin-outs.
8501   Same as 7501
8502   Used in C128s.
```

Some processors of the family which are not 100% compatible:

```
65C02  Extension of the 6502
65SC02 Small version of the 65C02 which lost a few opcodes again.
65CE02 Extension of the 65C02, used in the C65.
65C816 Extended 65C02 with new opcodes and 16 bit operation modes.
2A03   Nintendo NES/Famicom, modified 6502 with built-in sound controller.
```

About/Credits

About

This document is part of the no$sns emulator/debugger help text. Copyright 2012 by Martin Korth (nocash).
PPU and APU specs are widely based on Anomie's great specs (on that part I've mainly rearranged and reformatted the Anomie's txt files, verified most of it on real hardware, and discovered only a few additional details).
For information on Add-Ons like Controllers or Cartridges with Coprocessors: There haven't been any all-in-one specs (not even a complete list of existing add-ons), so I needed to collect scattered fragments of information from hundreds of webpages (and do a lot of puzzling and reverse-engineering on my own).
The result should be a fairly complete document that covers almost everything about the SNES hardware. There are still thousands of missing details (marked by words like unknown, whatever, might, maybe, probably, or by question marks). Info on such details (or donated hardware for research purposes) would be very welcome!

Credits

```
- Anomie (first ever reasonable/detailed SNES specs)
- Boris (donated a SNES and SGB back in 1999)
- SNES Central (cartridge PCB photos/scans)
- SNES Development (http://wiki.superfamicom.org/)
- snes9x - open source emulator (info on undocumented coprocessors)
- http://www.datasheetarchive.com/
- byuu (coprocessor decapping, info on snes hw glitches, cart memory maps)
- Segher's weird and wonderful CIC (rev-engineered CIC opcodes)
- DogP (lots of hard to get info on NSS)
```

Plus many-many other people who released valueable bits of information on their own webpages, wikis, and in various forum threads.

Homepage

http://nocash.emubase.de/sns.htm - no$sns emulator/debugger updates
http://nocash.emubase.de/fullsnes.htm - snes specs updates (html version)
http://nocash.emubase.de/fullsnes.txt - snes specs updates (text version)
http://nocash.emubase.de/email.htm - contact

Index