Nintendo DS game console

For Architecture and Technologies for Entertainment Software by Märten-Anhtuan Tran-Tõnissoo

General description

Nintendo DS is dual-screen handheld game console with one screen being touch-screen that can be used with stylus or finger. Integrated WIFI (IEEE 802.11), dual speakers and microphone (and 2 cameras in DSi version) make this a very tempting platform for both users and developers as it offers more ways to communicate and control your games and offers a lot of challenges while doing that [2][4].

There are 3 different "generations" of DS. First is the version was released in 2004 and was code name Nitro or just DS. DS stands more commonly for Dual Screen but it has been used as Development System. In 2006 the DS Lite was released. In October 2008 the new DSi was announced and released in Japan [4].

Here are some pictures of these consoles.

Original DS

DS Lite

DSi

The work here is mainly focused on the Nintendo DS Lite as the details about the new DSi are bit scarce and unreliable. The hardware architecture is basically the same in all three consoles. The later only get smaller and add additional functionality and tweak the existing system.

DS has both official accessories (different colour consoles for example, rumble pack, headset) and third-party add-ons (memory cards, booting accessories, rumble packs).

DS uses a 3 inch touch screen as additional user input to the standard D-pad and four buttons. It can detect only single point or average of multiple touches [4]. Also it features stereo speakers (a 16 hardware channel sound output) and microphone. Both screens are TFT LCD and have a resolution of 256 x 192 pixels (dimensions of 62 x 46 mm and 77 mm diagonal, and a dot pitch of 0.24 mm) [4].

Nintendo DS Lite weights 218 g and its dimensions are 133 mm × 73.9 mm × 21.5 mm. The processors are the same in DS and Lite, but in Lite they are made on a smaller custom process so they require less power [6].

Battery in DS is 850mAh (Lite has 1000mAh) and lasts around 4-10 hours [16].

Position on the market

How popular is this console? If you look at the following figures it will give you a general idea of its success.

As of October 2008 DS and DS Lite have sold over 84 million consoles worldwide [8]. Due to its cheaper to make hardware than its competitors (PSP) and the well known name of Nintendo the console has a strong position in the console market and the release and sell out of its new console in Japan (200 000 consoles in 4 days!) [7] only solidifies it.

If you take into account that one DS Lite console costs between £50-100 and you add those numbers you get the idea how much Nintendo is making from this console.

The console is sold in America, Europe and Asia and is customized to particular market. This makes the console more popular among its users and uses the right strategy to sell it for that cultural background (not to mention the language differences among other things).

From these sales figures it a fair guess that users like this console. But there are always two sides for a story and the other side for this one is developers. How are developers taken on this console?

There are over 200 games that are officially released for Nintendo DS game console [24]. If you add to that all the games that are supported from GBA (until Nintendo DSi release, but more on that later) and all the homebrew software and games you get the idea how powerful this console is for developers.

What is offered on DS ?

Nintendo DS has resolution of 256 x 192 on 2 screens. This provides a lot of space for games on a mobile console.

It has a wide range of single player games and it also supports 2-16 player multiplayer games.

Due to the nature of this console there is no support for MMO games.

The platform is capable of basically any single player genre type. For example of available genres there are:

- Adventure
  - Final Fantasy
  - Harry Potter
  - Lego Star Wars
- Action
  - Bomberman
  - Kirby
  - Mario vs. Donkey Kong
- First-person shooter
  - Metroid Prime Hunters
  - Call of Duty
- Puzzle
  - Sudoku
  - Actionloop
- RPG
  - Final Fantasy
- Sports
  - Mario Kart
- Lifestyle
  - Music – Electroplankton
  - Cooking – Cooking Mama
  - Language – Practice English!
- Party
  - Mario Party DS
- Strategy
  - Age of Empires
  - Advance Wars
- “Brain training games”
  - Big Brain Academy
  - Dr. Kawashima’s Brain Training
  - Professor Kageyama’s Maths Training

This list is not by far all that is offered but it gives an idea how many different games and software there is available and supported by this console.

More detailed look on hardware

CPU

Nintendo DS(i) uses 2 ARM processors.

First is ARM946E-S which runs at 67Mhz and is RISC 32 bit CPU. It has the power between 200 and 300 MIPS [5].

On should know that there are reports that the real 66MHz can be used only with cache and tcm, all other memory and I/O accesses are delayed to the 33MHz bus clock, that'd be still quite fast, but, there seems to be a hardware glitch that adds 3 waitcycles to all nonsequential accesses at the NDS9 side, which effectively drops its bus clock to about 8MHz, making it ways slower than the 33MHz NDS7 processor, it's even slower than the original 16MHz GBA processor [21].

Second (co-)processor is ARM7TDMI with clockrate of 33Mhz and 16/32-bit RISC[5].

Both ARM use the Little-Endian convention. [5]

ARM7 and ARM9 are often configured to use Thumb mode (16 bit mode), instead of the ARM mode (32 bit mode). The reason for this, in both cases, is memory space: ARM7 has limited program space, whereas ARM9 has limited instruction cache space. As for the 16 bit-wide bus, ARM (32-bit) instructions would saturate the bandwidth and seriously decrease overall performance.[5]

Neither CPU of the DS has a *Floating Point Unit* (FPU), so all floating-point operations must be emulated (not hardware support, they have to be done in software), which is very slow (so avoid by all means `float` and `double` variables). Most computations use therefore:

- fixed-point (integer) arithmetic for basic operations (addition, subtraction, multiplication and division
- (pre-computed) Look-Up Tables (LUT) for other computations, such as trigonometric ones (`cos`, `sin`, etc.).

As a consequence of having two CPU, a DS ROM has to include at least two executables, one for each of the ARM (not mentioning application-specific data). At runtime the executables will have to work simultaneously, which involves often having to be synchronized and to share data. [5]

An important developing rule is to prefer using the `CPU native words` (16-bit datatype if the CPU is in 16-bit mode, 32-bit datatype if the CPU is in 32-bit mode) for most of your calculations, loop index, etc.: then the bus, the CPU registers and the instruction set will be "inline", as efficient as obtainable, whereas smaller datatypes would lead to runtime performance penalties. Unless they are packed in a structure, they will not be any smaller. Using such words only requires careful casting, because an ARM CPU is quite picky about memory alignment. [5]

Internal (Working) RAM, or I(W)RAM, is the address range of RAM that is internal to the CPU. It acts similar to a cache, but is software addressable. This saves transistors and power, and is potentially much faster, but forces programmers to specifically allocate it in order to benefit. [22]

ARM are used because they are small, cheap and dont demant a lot of power. All of these features are very important in mobile devices.

ARM7 role

The full name of this processor is `ARM7TDMI`, meaning it is an ARM 7 core (a.k.a. ARM v4), which can read Thumb (16-bit) code, has a Debug mode and a fast Multiplier. On the DS it has neither an

instruction cache nor a data cache, but it is a bit compensated by the fast memory it owns, the 64-kilobyte IWRAM linked with a 32-bit wide bus. [5]

There are also two 16-kilobyte WRAM banks that can be assigned independently to the ARM7 or ARM9, with a 32-bit wide bus in both cases. The two ARMs cannot access these banks at the same time. Commonly, both banks will be mapped to the ARM7 (devkitARM defaults): as they form then a continuous block with the ARM7 IWRAM, this processor is effectively given 96 kilobytes of fast memory. [5]

The ARM7 is the only CPU that can be used for controlling the touchscreen. Most applications use boilerplate code that sets up an interrupt handler for the already mentioned *Vertical Blank Interrupt* (VBI). Not for rendering purpose here, but for synchronization, so that the interrupt handler dedicated to ARM7 input reporting can be scheduled regularly. The ARM7 boilerplate code gets the value of the touchscreen parameters and stores them in a data structure the ARM9 can access to. [5]

The ARM7 is also the only CPU that can make use of the microphone, the sound playback, the wireless communications and the real-time clock. [5]

Depending on a data being const or not, the linker will place that variable respectively either on an average memory rather unconstrained or in IWRAM, which is the fastest but one of the smallest. Therefore each time one forgets to specify the `const` qualifier for an actual constant, it may use unnecessarily the most researched IWRAM instead of low-end memory. [5]

The free toolchain devkitPro includes a default ARM7 program to handle basic tasks like managing interrupts, reading the touchscreen, the microphone and the realtime clock, performing very simple sound playback, etc. [5]

The latest devkitARM linkscripts and the default ARM7 core reserve the switchable IWRAM for ARM7 exclusive use. Nintendo official code also does this. [5]

The ARM7 processor supports both 32-bit and 16-bit instructions via respectively the ARM and Thumb instruction sets (the later is a subset of the former). 16-bit code is smaller than 32-bit one and, in some specific cases, faster (usually slower though). 16-bit code is often preferred, as it has more chances to fit in the tiny instruction caches. Moreover the size of ARM7 code must be in general 64 kilobytes or smaller. [5]

ARM9 role

Due to its superior power compared to the ARM7, the ARM9 is the main processor and as such will take in charge most of the work. Most of application-specific code is expected to run on it.

The ARM9 uses two additional built-in caches (beyond the usual CPU L1 caches): one for the instructions (ITCM, 32 kilobytes), the other for the data (DTCM, 16 kilobytes). Each is accessed thanks to a dedicated 32-bit wide bus. Both are caching accesses to the main memory and increase the ARM9 performances a lot, at the expense of a small additional level of complexity: as neither the ARM7 nor the DMA circuits are aware of these two caches, care must be taken not to create inconsistencies with their view and the one of the ARM9.

To make a better use of these caches, various primitives are provided to ensure they stay in sync with the main memory. This includes a mirror of main memory that is not cacheable (`02400000-027FFFFF`), and a way of flushing the data cache (`DC_FlushAll`) [22].

ARM9 `deals usually with :`

1. an **initialization** stage, to set screen modes, memory banks, interrupt handlers, to initialize various libraries (ex: PAlib, libfat, dswifi, etc.) and to perform as many tasks as possible (ex: loading the resources from mass storage, setting up Wifi connections, etc.)
2. a **main loop**, in charge of:
    - **game logic**, including any AI algorithms
    - **input management**, partly read from the ARM7 (touchscreen, some keys)
    - **audio and video rendering**, partly managed by the 2D/3D engines too
    - **I/O exchanges** with a mass storage device, if any
3. a **shutdown** stage, to stop all subsystems properly, including flushing write buffers (closing files, unmounting mass storage, etc.), stopping the Wifi connections, etc.

In GBA mode the ARM9 is not powered, only the ARM7 can be used.

[5]

Interrupts

An interrupt (or IRQ, for *Interrupt Request*) is a way for a CPU to stop immediately the current execution path in order to run another function, called an interrupt handler, instead. Hence when a hardware or software interrupt occurs, the processor saves first some information so that it knows where to go back and in which state, then the handler associated to this interrupt for that CPU is called at once and, when that function returns, this CPU continues executing the piece of code it was executing before it was interrupted, as if nothing had happened.

Therefore interrupts allow to perform tasks that should not wait, either because otherwise they might be missed (ex: a keypress described in a hardware register) or because the application must react directly to them (ex: a Vertical Blank Interrupt that would trigger rendering, see below).

Interrupts allow also to use the CPU sleep mode, since they provide a way of waking it up as soon as it becomes needed again, thanks to a BIOS routine. In this low power mode, the ARM9 stops processing instructions and powers down some memory banks to save battery charge. There is an ARM instruction named SWI (for *SoftWare Interrupt*) with one numerical parameter that means: enter the SWI handler and pass that value in order to know which interrupt or combination of interrupts the CPU should be waiting for, from now on.

Helper libraries, including libnds, offer higher-level interrupt management. If most of the program is in interrupt handlers, then the CPU may sleep most of the time, preserving the charge of the DS batteries. Note that interrupt handlers are meant to be executed in a short time though, as otherwise they might be interrupted themselves. Too many cascading long interrupts might prevent the DS from ever returning to the main interrupted code.

By default when an interrupt is triggered, the CPU jumps to the standard BIOS interrupt routine, a function which is stored in a special memory address that we can write to. By storing here a pointer to one of our own functions, we can cause the interrupt to be processed by an user-specified function, i.e. we can define our custom interrupt (IRQ) handler.

Hardware-interupts

The DS supports 23 different hardware interrupts, named here according to the libnds convention (unless specified otherwise, these interrupts are available both for the ARM7 and the ARM9):

- **screen-related**: triggered when an horizontal blank, a vertical one occurred, or when a user-specified number of scanlines (set in `DISPSTAT`, compared to `REG_VCOUNT`) were displayed (`IRQ_HBLANK` / `IRQ_VBLANK` / `IRQ_VCOUNT`)

- **timer-related**: triggered when a timer overflows (`IRQ_TIMER0 / IRQ_TIMER1 / IRQ_TIMER2 / IRQ_TIMER3`)
- **network-related:** : triggered when (supposedly) a generic network or Wifi-specific event occurred (`IRQ_NETWORK / IRQ_WIFI`) [ARM7-specific]
- **DMA-related**: triggered when a DMA transfer is over (`IRQ_DMA0 / IRQ_DMA1 / IRQ_DMA2 / IRQ_DMA3`)
- **input-related**: triggered when keys being pressed match a user-specified mask, `REG_KEYCNT` (`IRQ_KEYS`)
- **card-related**: triggered when an event occurred in the GBA port, when a data transfer for the DS card is completed, or in another event occurred on the DS card (`IRQ_CART / IRQ_CARD / IRQ_CARD_LINE`)
- **IPC and FIFO-related**: triggered when the IPC is synchronized, when receive FIFO is not empty, or when send FIFO is not empty (`IRQ_IPC_SYNC / IRQ_FIFO_EMPTY / IRQ_FIFO_NOT_EMPTY`)
- **render-related**: triggered when an event occurred in the geometry engine (`IRQ_GEOMETRY_FIFO`) [ARM9 only]
- **SPI-related**: triggered when an event occurred in the SPI bus (`IRQ_SPI`) [ARM7 only]
- **lid-related**: triggered when the lid state changed (`IRQ_LID`) [ARM7 only]

Once the IRQ subsystem has been initialized (`irqInit`), handlers can be associated to interrupts (`irqSet`, `irqClear`). libnds provides a default overall interrupt dispatcher that can be overriden (`irqInitHandler`).

Interrupts can be enabled/disabled separately (using `REG_IE`, or `irqEnable`, `irqDisable`), and temporarily disabled as a whole (`REG_IME`).

The ARM7 stores the addresses of its interrupt handlers in a hardcoded memory location, whereas the ARM9 defines these addresses relativey to the DTCM.

Both CPU can trigger interrupts to each other (if the ARM9 allows it). It is convenient to send a notification to the other CPU when, for example, there is data for it waiting to be read on a shared area in RAM, once the calling CPU has finished filling it.

Screen-related interrupts: Horizontal & Vertical Blank Interrupts

The DS screens are updated neither as a whole nor permanently: the graphics hardware draws pixels one by one, from the top left to the bottom right of each screen, line by line. It then waits for a while (a fixed duration) before starting to draw again.

If the framebuffer (the place in memory where the pixels are stored) is modified during the redrawing process, the user may see visual artefacts in the form of partly-updated images, the top-left part being rendered according to the previous state of the framebuffer, bottom-right with the current one.

During a short duration after a line is drawn, during a longer one once a full screen is rendered, the hardware remains idle. These moments can be used to perform safe rendering: no partial redraw is to be feared then.

Two special interrupts are regularly fired, so that the programs can use these two favorable periods: one, the *Horizontal Blank Interrupt*, occurs whenever a line has been rendered. Your program can use this first idle duration to perform rendering operations. The other one, more famous, is the *Vertical Blank Interrupt* (VBI), that is fired once a full screen has been redrawn. Your program should use this longer duration to perform at least framebuffer-related operations, while the hardware moves from the last line back up to the first line. This idle stage is called *Vblank*, as opposed to *Vdraw*, the screen refresh time.

At both screens are refreshed at 60 Hz, the period between two VBI is 16,7 ms long. The VBI is called also *vsync*, since it allows for *vertical synchronisation*.

Note that all other operations (input reading, sound output, application logic, etc.) can be performed regardless of these two interrupts. But the VBI, beyond its use to avoid visual artefacts, provides too a hard real-time 60Hz time-base. This time base can be used to schedule operations on a regular basis. Timers are useful for that task too.

Key Interrupts

Keys can be read thanks to several methods, including the interrupt-based one. In this case a specific IRQ handler is registered. This handler will be triggered indeed when a key is pressed, but it will not be called when the key is released, which reduces quite a lot the interest of this method for key handling.

Beyond screens and sometimes keys, FIFO, IPC and timers make heavy use of interrupts.

Software Interrupts

These SWI (for `SoftWare Interrupt`) are triggered by the program itself, thanks to the ARM instruction named `swi`, and result in a DS BIOS function being called.

One would prefer to use pre-made encapsulations for these BIOS calls, for example the ones provided by libnds (ex: `swiSoftReset`).

The DS supports 25 different software interrupts.

[5]

Inter-Process Communications

As each ARM has specific abilities (ex: the ARM7 is the only one that can access the hardware for sound and wireless), they have to communicate one way or another to send to the other ARM commands to be executed on the sender behalf.

The ARM CPU can communicate thanks to IPC (*Inter-Process Communications*) based on a set of registers managed thanks to a (possibily bidirectional) FIFO (*First In, First Out*) data structure. It corresponds actually to message queues with an asynchronous communication protocol.

Communication between ARMs is tricky: beyond the classical issues of synchronization of the concurrent accesses (parallelism between the ARMs), one has to keep in mind the ARM7 is not aware of the ARM9 data cache (DTCM), which may lead to inconsistencies if using the main memory to share data.

One solution is:

1. to allocate memory **from the ARM9**: with the default link script, any allocation done from the ARM7 would return a block in the ARM7 private memory, not addressable from the ARM9
2. **and** to ensure the allocated memory is out of the ARM9 data cache (otherwise, if not flushed, the ARM7 may see non-updated memory) by using, on this processor, the **non-cached main memory mirror** (`0x02400000-0x027fffff`, add `0x400000` to the normal main RAM address to get to the uncached mirror), even if thus the ARM9 access to the data will be slower. An alternative solution to the mirror would be to flush/invalidate the cache manually, although this approach does not seem 100% reliable

There are ways to use Custom-made IPC but they are not recomended and better solution would be to use Hardware-based IPC which DS has provided.

[5]

Data transfers

Programming the DS involves performing numerous data transfers, for graphics, sounds, application data, etc., from various regions in address space to various other regions (memory banks, slot-1, slot-2, IWRAM, etc.).

These transfers can be achieved thanks to various means, each with its own forces and weaknesses. Starting from the most often favoured transfer method to the least in case of a significant transfer:

- **(asynchronous) DMA transfers**:

  although "only" up to four of them can run simultaneously, DMA transfers (open to both ARM, ex: `dmaCopy`) are interesting because they offset this load from a CPU. They run in background and trigger an interrupt when having finished, letting the CPU perform other tasks in the mean time. There are not necessarily the fastest of all transfer methods, but this is more than compensated by their parallel execution feature.

  A drawback is that, as long as the DMA transfer is running, both CPU will be locked off the bus to the main RAM, to prevent the CPU and the DMA controller from trying to access the bus at the same time, causing a collision.

  Thus the ARM9 should execute from its "second-level" instruction cache (ITCM) reading/writing data from/to its "second-level" data cache (DTCM), otherwise it will be frozen, waiting for the bus. `dmaCopy` cannot access the DTCM region of the ARM9, which is where the stack is placed. Thus if the source of a DMA transfer is the main RAM from the ARM9, the DTCM must be flushed beforehand. Some interference between DMA transfers and interrupt handling have been reported, when in doubt swap to **memcpy** to see whether it improves stability.

  As for the ARM7, apparently, even when executing from its IWRAM, it will be frozen

- **memcpy**: simple and usual method, yet rather efficient (see also memset for setting instead of copying)
- **basic 'for' loop**: little use for that, as previous methods are faster and offer at least the same features anyway
- **swiFastCopy**: this libnds-provided call branches to a DS BIOS routine. According to that source, should be ruled out due to a bug leading to poor performance. Transfers more quickly than swiCopy, but has higher interrupt latency
- **swiCopy**: as swiFastCopy, suffers from a bug apparently

[5]

GPU

Video RAM (VRAM) is divided into 9 banks that have different uses and can be combined with each other (from 16 – 128 KiB, 656KiB in total) [20]. Its important to know that depending on the video mode, each rendering core will retrieve its video-related informations (ex: bitmaps, sprites, tiles, textures, maps) from hardcoded regions defined in overall memory address space. It accepts writes in 16 or 32-bit.[5]

The ARM946E-S CPU processes 3D rendering and the ARM7TDMI processes 2D rendering for DS games and Game boy Advance gameplay [4].

The system's 3D hardware can perform :

- **transform and lighting** - performs 3D space to screen conversion, including lighting. Having it hardware accelerated frees the CPU from this computation-intensive task.
- **texture-coordinate transformation** - textures (images used to wrap 3D shapes) can be used.
- **texture mapping** - applies a texture to a shape, here without blending.
- **alpha blending** - combines an image with a background to create the appearance of partial transparency.
- **anti-aliasing** - minimizes the distortion artifacts due to on-screen rendering (enhances the resulting image). Here available for edge drawings, not full-screen.
- **cel shading** (a.k.a. toon shading) - simulates hand-drawn (like cartoon or comic books) computer-generated graphics
- **z-buffering -** management of image depth coordinates, one of the solutions to the visibility problem. Hardware fog is available.

[5]

It uses point (nearest neighbor) texture filtering, leading to some titles having a blocky appearance. [4]

The system is theoretically capable of rendering about 120,000 triangles per second at 60 frames per second. Unlike most 3D hardware, it has a set limit on the number of triangles it can render as part of a single scene; the maximum amount is about 6144 vertices, or 2048 triangles per frame (this is for 1 screen).[4]

The 3D hardware is designed to render to a single screen at a time, so rendering 3D to both screens is difficult and decreases performance significantly. The DS is generally more limited by its polygon budget than by its pixel fill rate.[4]

There are also 512 kilobytes of texture memory, and the maximum texture size is 1024x1024 pixels.[4]

3D core behaves alot like an OpenGL state machine, allowing for wrappers and the reuse of rendering code and data. [5]

The system has two 2D engines (one per screen). These are similar to (but more powerful than) the Game Boy Advance's single 2D engine; however, the cores are divided into the *main core* and *sub core*. Only the main core is capable of vertex 3D rendering. 2D supports maximum of 128 sprites per screen (hardware limit in 2D core built in memory).[4] Each entry is made of four 16-bit attributes, storing the size, shape and location of the associated sprite. Up to 32 out of the 128 entries can correspond to affine transformations (named *rotsets*), whose additional attributes specify rotation and scale. Hence up to 32 rotsets can be defined, but more than one sprite can be associated to a given rotset. [5]

The VRAM banks previously mentioned are to be mapped according to the expected layout for 2D memory, which is mostly made of:

- **background** memory, which contains either tilesets (32 blocks, 16 kilobytes each) and tile maps (32 blocks), or bitmaps
- **sprite** memory, which contains tiles for each sprite

[5]

There are 8 graphic modes for main screen and 6 for sub screen. Each graphic mode can have one, two or four background. They are layers that you hold your images, animated characters, text on. DS has eight background Control Registres for each background on main ans sub screen [20]. There are

special registers for both screen and a display capture control register. More info on those can be found on DSWiki Graphic modes[20] page.

If the rendering takes place while the screens are redrawn, then the user will see on its screens images partly updated, leading to unwanted visual artefacts. The solution is either to modify the screen content only between two redraws or to use page flipping. Page flipping is a method that consists on rendering in a screen buffer while the hardware, simultaneously, displays another buffer. At each VBI, buffers are exchanged so that both tasks can continue.[5]

Memory

Main memory is 4 MiB and is shared by two processors (only one processor can access it at one time) [16]. Nearly all the game data and ARM9 executable are in main memory. This memory is rather slow (compared to ARM9 caches for example). ARM7 code is usually in IWRAM (for *Internal Working RAM*, 64 kilobytes of fast RAM, 32-bit wide, that only the ARM7 can access), that is much faster than keeping it in main memory with ARM9 data. [5]

There are two memory banks of *Tightly Coupled Memory* (TCM) is the ARM9. They are high-speed memory, directly contained in the ARM CPU core.[5]

The DTCM, for *Data Tightly Coupled Memory*, is a special 16-kilobyte memory area in the ARM9 which can be mapped to reside at various actual physical addresses. It is a lot faster than the main RAM, therefore the standard ARM9 linkscript places its stack in DTCM. [5]

As for the ITCM, for *Instruction Tightly Coupled Memory*, it is a special 32-kilobyte memory area in the ARM9 which can be mapped to reside at various actual physical addresses. It is a lot faster than the main RAM, so it should be used for small (preferably 32-bit) functions that are computation-intensive and/or frequently called. For example, libnds uses that region to store the interrupt dispatcher. [5]

Although these two tightly coupled memories (TCM) are faster memories than RAM, are internal to the ARM9, and are used for storing high performance code/data, they are actually completely separate areas of memory than the instruction and data (L1) caches of the ARM9. [5]

A minor disadvantage is that TCM cannot be accessed by DMA. The main advantage is that, when using TCM, the CPU can be kept running without any waitstates even while the bus is used for DMA transfers.[21]

So from the ARM9 point of view, the memory hierarchy is, from closest/fastest/smallest to farest/slowest/biggest: ARM9 caches (L1) > ARM9 TCM caches ("L2") > RAM and other memories. [5]

Both ARM9 TCM seem to rely on 32-byte cache lines. As a cache line cannot be partially read or written apparently, special care must be taken when invalidating or flushing them. [5]

Miscellaneous information about the hardware

It also has real-time 33Mhz clock, two slots for FLASH cards (solid state ROM game cards with flash memory or EEPROM for save data, cards are 2 gigabit in size.) and flash-able firmware. [5]

Much of the DS programming is done controlling hardware memory-mapped registers to controll the system features. Most useful registers are defined by the low level library named libnds, to manipule abstract plain names (ex: `DISPLAY_CR`) instead of raw addresses (ex: `0x4000000`).[5]

Game Logic

These are real issues in the PC world where no two hardware platforms behave the same and where uncoupling all subtasks (audio/video rendering, AI, input reading, network management, etc.) becomes soon tricky, but in the console world the situation is somewhat simpler: everything ought to be hardwired and paced according to the same clock, here the 60 Hz rhythm provided by the VBI.

Instead of putting that logic code is the VBI handler, the best way is to keep the `while()` loop in the `main()` function, but to make each iteration end with a request for the DS to sleep until next interrupt occurs (with libnds, one would use `swiWaitForVBlank()`). Therefore both rendering and logic will run at a predictable 60 Hz, and thanks to the sleeps the DS will exhaust its batteries later.

MMU

The DS does not have a Memory Management Unit (MMU), a hardware component responsible for handling memory accesses requested by the CPU, notably to enable virtual memory management, memory protection, cache control, bus arbitration or bank switching.

It reduces the potential stability of operating systems (as Linux; if they are to be used on a DS), and prevents to access, as fallback swap, to mass storage that would be available in slot-2-provided removable media.

Non rendering-related hardware accelerations

The DS contains both divide and square root accelerators.

Serial Peripheral Interface

Various subsystems (power management, firmware, touchscreen, sound volume and control, microphone amplifier control and gain control, back-lighting of screens, power LED, battery status) can be accessed only thanks to the SPI (*Serial Peripheral Interface*) bus.

DMA

A DMA (*Direct Memory Access*) copy is basically a fast, CPU efficient, hardware accelerated copy: it allows certain hardware subsystems within the DS to access system memory for reading and/or writing independently of the CPU.

There are four prioritized DMA channels. They can transfer data asynchronously from the main CPU: the ARM9 initiates the transfer and, if it does not access to main memory, can continue its work while the DMA transfer is going on. Then it will receive an interrupt from the DMA controller once the operation has been done. It results in data transfer with much less CPU overhead.

For each DMA channel, the status (enabled/disabled), the start of transfer (immediately/at next HBL/at next VBL), the size of each atomic transfer (16/32 bits), the trigger of an interrupt when the transfer is over, the source and destination transfer pattern (fixed/ascending/descending) can be chosen. All these informations are set thanks to a register (`DMA_CR` in libnds) that tells too whether a given DMA channel is busy (`dmaBusy` in libnds).

Note though that during a DMA transfer the CPU is locked off the bus, thus restraining a lot what it can done in the mean time. Note also that the DMA circuits are not aware of the ARM9 caches, leading to potential inconsistencies.

Timers

A timer is a hardware function that can be set to raise an interrupt at regular intervals, once enabled. There are eight incrementing 16bit timers, four for each CPU.

Timers are useful to wait for a given duration, or to schedule an action regularly. There are four timers in the DS, each running at up to 33.514 MHz. This frequency can be scaled down of various powers of two, and timers can cascade: then for example timer #1 would increase only on timer #0 overflow. If needed, a timer may trigger an interrupt when it overflows.

Timers can be managed thanks to two registers, named in libnds TIMER_CR(x) and TIMER_DATA(x) with x in 0..3.

[5]

Network capabilities

From a network point of view the Nintendo DS has 2 types of wireless connections.

First is a special wireless format created by Nintendo and secured using RSA security signing (used by the wireless drawing and chatting program PictoChat for the DS) [4].

Second is a standard WIFI (IEEE 802.11) connection for network play. It can connect to a Wi-Fi router or straight to a nearby console.

Download play is available where multiplayer games can be played with one game card. Game is stored in the console until it is turned off. Download stations at shops provide game demos via this method.

PictoChat is also available on local wireless range where users can communicate and send picture messages [3].

The new DSi will use similar structure as Nintendo's Wii Channels to deliver game demos, Photo Channel and web browser and later on downloadable full games [11].

Depending on the game it can be played locally or though a router via internet worldwide.

Additional add-ons support Opera browser and homebrew software can add more network functionality.

Software development

Software for DS can be developed as a Nintendo acknowledged developer or as homebrew software.

The homebrew software can be developed by anyone via public knowledge and freely available software. There are sites that give information how to develop for this console (such sites as NDSTech or a very extensive guide to homebrew DS development [5]).

One available toolkit is DevkitPro with is based on C/C++ language that makes it accessible to many programmers due to standard well used programming language. There are also many libraries available (dswifi, LibFATDragon, LibNiFi and many more). [15]

However this is troublesome for bigger development projects, for beginners and isn't supported by Nintendo.

Software can be ported to the console via Gameboy Advance port or DS game card port. Both solutions will require special hardware to run your programs that make additional requirements to the developers. On the second hand these hardware are not very expensive [4].

To get Nintendo's development tools you have to be accepted to Nintendo's official developer support program.

To be accepted into Nintendo's official developer support program, companies must have a game development team and adequate experience in certain areas. You need this to get access to Nintendo SDK. More information on getting Authorized Developer status can be found on Nintendo Software Development Support Group Home Page. Because of the NDA details about the tools available from Nintendo are very hard to find [9].

The “improvements” of DSi

Many are blinded by the all new features of DSi like new bigger screens, dual cameras and so on. But Nintendo has taken a few big steps back on the console with some not so good features.

New Nintendo DSi may suffer from battery loss due to new processor and audio hardware [1]. This is to be expected if the console is getting smaller and more powerful, but still it is not good for a mobile gaming console.

DSi game console is region locked which means that all DSi games will not run on any other console from other regions. Although your pictures and web browsing isn’t affected it limits the availability of the games and can cause many problems for people who travel a lot or live in regions that games aren’t available. Nintendo answered the questions about region lock with justification about parental control needs and service tailored to specific market. [17]

The new DSi has gone into extra trouble to disable any flash cards that allowed homebrew application to run. They justify this with usage of illegal ROMs with this technology. But it effectively stops all homebrew development as it is not possible to run them on the new console anymore. This has serious consequences for the new console and developers who can’t afford Nintendo’s license [18].

It also removes GBA slot that was used to give backward compatibility with GBA games. As many add-ons (rumble pack) use this port it will definitely lower the product popularity for many people [19].

Conclusion

The Nintendo DS is a very popular gaming console among users around the world. The developers have embraced it and published a wealth of games for this console.

The console hardware is not as powerful as other of its brothers (PSP), but it offers other features to make it a well rounded hardware capable of 2d and 3d graphics and doing it on two different screens. The additions of microphone, wireless connectivity and cameras provide more opportunities to make unique products for this console.

The later developments of Nintendo's latest console DSi are questioned by many, but as it hasn't been released worldwide it is not sure how well it will do. One will have to wait and see if the sceptics are right about some cut features and usefulness of others.

One big drawback in the consoles future is Nintendo's idea to limit homebrew development by making it harder to develop and run software that is not made with their licensed tools and developers. This can lead to choosing of other platforms to develop and if students and people wishing to study portable game console software development have to choose other platforms and later choose a work in relation to their knowledge – it all adds up to less people using the DS for developing new software.

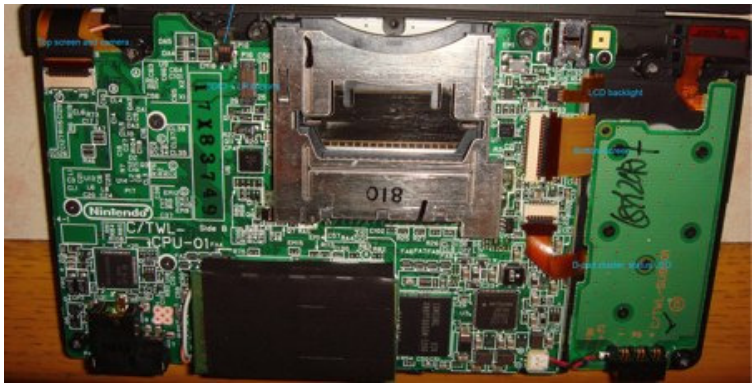
As people get more mobile and demand more from mobile devices it's a fair guess that Nintendo DS console will offer a solid platform for users to use and developers to develop on for the future.

Pictures

Picture 1. DSLinux[14]

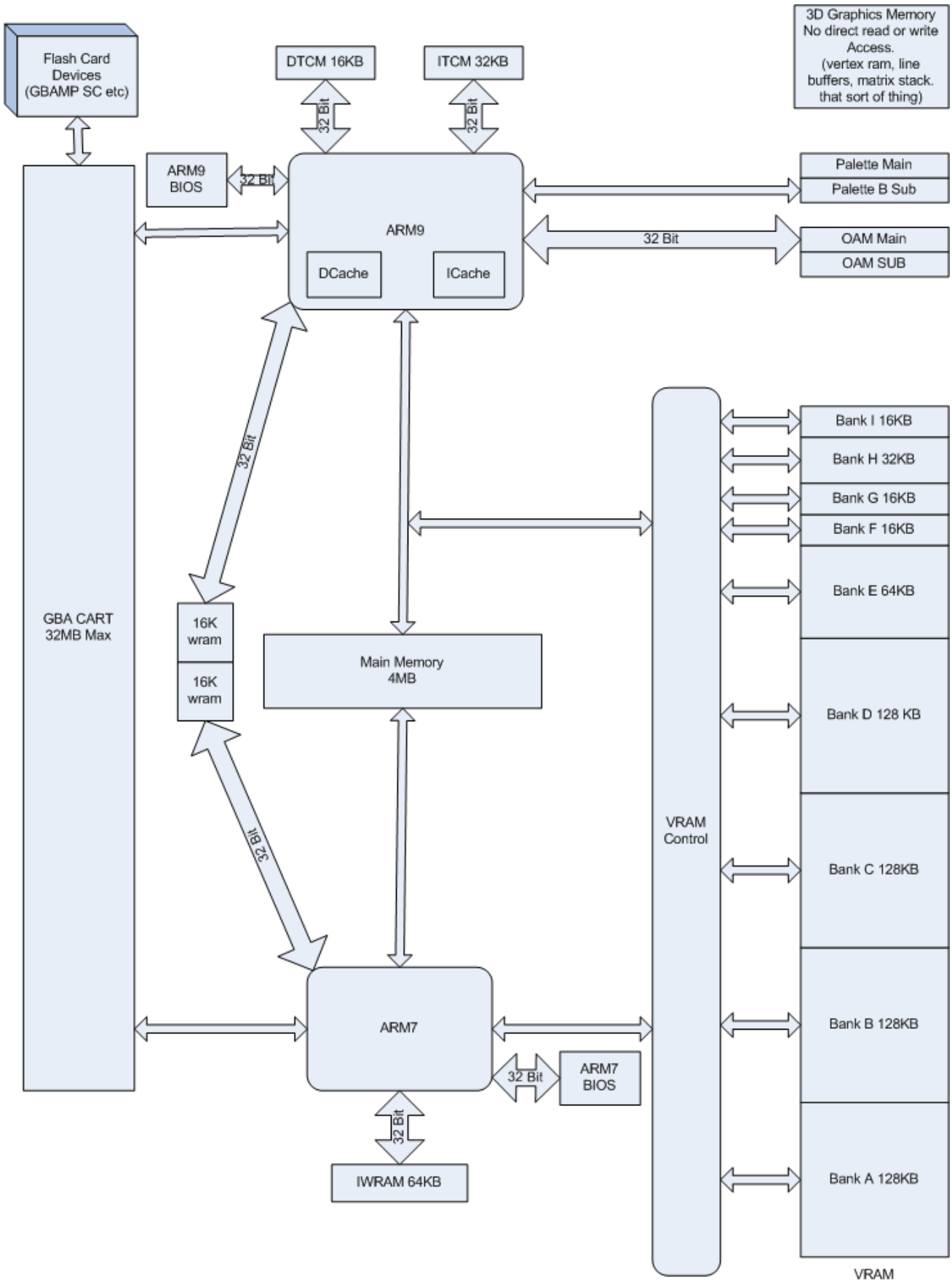
Picture 2. PointyRemote[13]

Picture 3. DS Lite opened[10]



Picture 4**.** New DSi[2]

Memory Layout of DS



If not stated otherwise, all busses are 16 bits. Diagram from [23]. Exact memory map can be found on Dev-Scene[23] page.

References

[1] http://gizmodo.com/5081197/deconstructed-dsi-reveals-beefier-processor-might-be-draining-battery-life
[2] http://nintendo-ds.dcemu.co.uk/nintendo-dsi-teardown-170453.html
[3] http://en.wikipedia.org/wiki/ARM7TDMI
[4] http://en.wikipedia.org/wiki/Nintendo_DS
[5] http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html
[6] http://en.wikipedia.org/wiki/Nintendo_DS_Lite
[7] http://www.newstin.co.uk/tag/uk/85952595
[8] http://www.nintendo.co.jp/ir/pdf/2008/081030e.pdf#page=11
[9] http://www.warioworld.com/apply/
[10] http://kotaku.com/gaming/ds-lite/new-pics-of-ds-lite-compared-cracked-open-155140.php
[11] http://www.gaj-it.com/index.php/2008/10/02/nintendo-dsi-the-next-generation-of-ds/
[12] http://www.nintendo.com/ds/what/features
[13] http://www.1emulation.com/forums/index.php?showtopic=16859
[14] http://superbad.net/random/ds_hardware/ds_lite_with_slot-2_device_running_dslinux.jpg
[15] http://www.tobw.net/dswiki/index.php?title=How_to_start_coding
[16] http://www.tobw.net/dswiki/index.php?title=Ds_hardware
[17] http://www.computerandvideogames.com/article.php?id=198728
[18] http://uk.gizmodo.com/2008/11/06/nintendo_dsi_built_to_reject_r.html
[19] http://n-europe.com/news.php?nid=12393
[20] http://www.tobw.net/dswiki/index.php?title=Graphic_modes
[21] http://nocash.emubase.de/gbatek.htm#dstechnicaldata
[22] http://en.wikipedia.org/wiki/Internal_RAM)
[23] http://www.dev-scene.com/NDS/Tutorials_Day_2
[24] http://www.nintendo.co.uk/NOE/en_GB/games/nintendo_ds_games_1957.html