

# Advanced Programming Topics for Nintendo DS

Mark Jawad

Senior Software Engineer

Software Development Support Group

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Topics

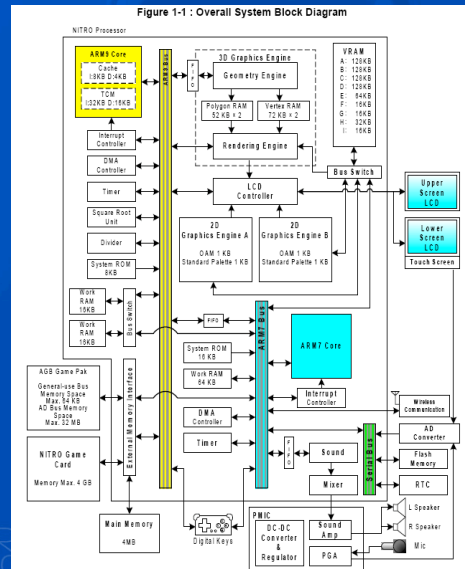
- Review of DS system architecture
- Role of the ARM7
- Role of the ARM9
- ARM9 arch. review
- Code Gen review
- Bus review
- Caches and TCM
- Implications of what we've learned so far
- Rules of THUMB
- Main Memory Display Mode (+DMA)
- Card DMA
- Interrupt processing and best-practices
- Fast data uploading during V-Blank
- Asynchronous processing

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Review of DS system architecture



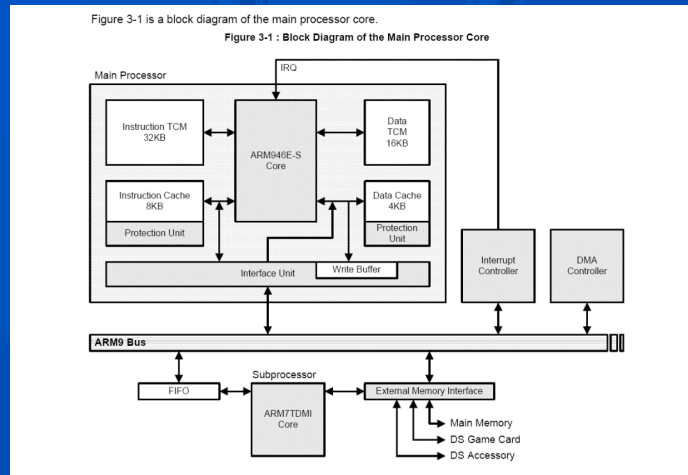
Point out:

## ARM9 + associated pieces

## ARM7 + attached peripherals

## Work Ram, and how it's attached

# Review of DS system architecture



DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo



# Role of the ARM7

The "subprocessor"

- Intended for handling bulk of IO work
  - Thus offloading repetitive tasks from the ARM9
- Can access hardware that ARM9 cannot
  - Wireless, sound chip, power management, touchpad, microphone, RTC, NVRAM (user settings), blinky lights for system status, etc
- ARM7TDMI
  - Von Neumann architecture
  - Not as heavily pipelined as ARM9
  - Has small Work RAM onboard
- Uses PXI protocol to talk to ARM9

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

ARM7TDMI is the same kind of chip used in the GBA; makes sense given the GBA-compatible mode that DS has.

This is how we achieve hardware-level compatibility with the GBA.

Note that on a Von Neumann machine, there's only 1 s

# Operation of ARM7

- Basic program operation
  - Creates a bunch of interrupt handlers
  - Drops into an idle loop
  - Interrupt handlers takes care of most needs
    - PXI requests, external device updates, etc
    - Some requests are too heavy-weight to do in an interrupt handler. These are sent to threads.
  - Threads handle "long-term" tasks
    - For audio, wireless, and other complex needs
    - Thread schedules may cause delays

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Notes on ARM7

- Doesn't have a cache
- Does have onboard RAM
  - Has 64K of internal Work RAM
  - Reserves all of the ARM7 / ARM9 “shared” Work Ram space for exclusive use (32K)
- Has priority access to main RAM!
  - Due mainly to Wireless and Audio needs
  - This *will* impact your game
    - But the amount depends on features used

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM7 memory access patterns

- Most code / data lives in ARM7 dedicated memory and shared work ram
  - Not in main memory
  - Very few code fetches from main memory
  - Data transmissions to/from main memory are occasional
    - Auto-sampled data (TP, MIC)
    - Audio data (samples, etc)
- Wireless complicates the picture a little

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

Need to discuss bus access pattern for all parts of the system here.

Audio data is transferred to FIFO by sound DMA. 32 bytes of data is transferred at the first time, then 16 bytes thereafter.

For Microphone and touch panel, one or two u16 word(s) written at once during the sampling.

Goes out to ram to write things like the X/Y button values, and RTC

# ARM7 SDK Components

- "Mongoose" component
  - Wireless code / data must be fetched from Main Memory\*
  - So more traffic to main mem when wireless is active
    - Remember, no cache on ARM7
- "Ichneumon" component
  - Wireless code / data is fetched from VRAM\*
  - But locks VRAM banks C, D

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

- Actually, the connection setup and teardown code is fetched from main memory.
- The code that operates while the connection is established is within the dedicated work ram + shared work ram

# Role of the ARM9

The "main processor"

- Reserved for your game code
- Mostly under your control
  - Some SDK pieces may be doing things you weren't aware of...
    - Threading
    - TCM use
    - Blocking on PXI command results
  - ..but we give you SDK source so you *do* have complete understanding

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Architecture Review

- See slides from previous DevConf for more detailed info on features, instructions
- Things worth noting:
  - DSP instructions can be beneficial, but are only accessible via ARM assembly code
  - Put data into local vars wherever possible
  - PLD (data preload) instruction is ***ignored***
  - Instruction Cache preload is ***supported***

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Code Generation

- Compiler places literal pools after each function
  - Any function that needs data other than the incoming parameters
- What's a literal?
  - Roughly: any data larger than a byte
    - Which means pretty much everything
    - Some literals can be placed in the actual instruction opcode (rare)
    - Most things (u32 and smaller) are stored in the literal pool
    - For larger data, lit pool contains the *address* of the data
  - Also: Addresses of any static or global variables

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

Lit pools mean more time on the bus (2x from normal code-only/data-only lines)



# ARM9 Code Generation

- Implications
  - There's at least 1 cache line with code + data overlap
  - No L2 cache means that the I-cache will need to load the line from RAM, and the D-cache will also load the line from RAM.
  - Can be detrimental to performance-critical functions

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Code Generation

- The more globals / statics accessed, the larger the literal pool
  - And you can see where that leads: program bloat
- Loading a global, static, et al takes 2 loads
  - 1<sup>st</sup> one is PC-relative, gets the addr of the global from the lit pool.
  - 2<sup>nd</sup> one actually retrieves the data
- Pack static or global variables into a structure to avoid the hit

Disassemble1

ARM9 Address: RecursiveEnumDir Move Reference Symbols...

```

02000FB4 EBFFFFDF BL 0x02000F38 <RecursiveEnumDir>
02000FB8 EAO00004 B 0x02000FD0
02000FBC E1A00006 MOV R0, R6
02000FC0 E1A02005 MOV R2, R5
02000FC4 E2881004 ADD R1, R8, #0x4
02000FC8 E2873014 ADD R3, R7, #0x14
02000FCC EB000212 BL 0x0200181C <OS_Printf>
02000FD0 E1A00004 MOV R0, R4
02000FD4 E1A01007 MOV R1, R7
02000FD8 EB00150E BL 0x02006418 <FS_ReadDir>
02000FDC E3500000 CMP R0, #0x0
02000FE0 EATFFFA0 BNE 0x02000FA0
02000FE4 E2800040 ADD SP, SP, #0x48
02000FE8 E8B00170 LDHIA SP!, {R4, R5, R6, R7, R8, PC}
02000FEC 0200089C <DATA> 0x0200089C (33609884)
02000FF0 020008A4 <DATA> 0x020008A4 (33609892)
02000FF4 020008A8 <DATA> 0x020008A8 (33609896)
InitializeAllocateSystem
02000FF8 E2800040 STMDB SP!, {R4, LR}
02000FFC E2800000 MOV R0, #0x0
02001000 EB000846 BL 0x02003120 <OS_GetArenaHi>
02001004 E1A04000 MOV R4, R0
02001008 E3A00000 MOV R0, #0x0
0200100C EB000848 BL 0x02003134 <OS_GetArenaLo>
02001010 E1A01000 MOV R1, R0
02001014 E1A02004 MOV R2, R4
02001018 E3A00000 MOV R0, #0x0
0200101C E3A03001 MOV R3, #0x1
02001020 EB000914 BL 0x02003478 <OS_InitAlloc>
02001024 E1A01000 MOV R1, R0
02001028 E3A00000 MOV R0, #0x0
0200102C EB0008AD BL 0x020032E8 <OS_SetArenaLo>
02001030 E3A00000 MOV R0, #0x0
02001034 EB000839 BL 0x02003120 <OS_GetArenaHi>
02001038 E1A04000 MOV R4, R0

```

Literal pool in this case points to 3 different / unique strings. So, can't compact it via a single structure..

Well, you could, but you have to go out of your way...

# ARM9 Code Generation

- Many ARM opcodes can have constant data of 8 significant bits or less
  - Bypasses the literal pool
  - Doesn't necessarily mean that you're limited to a single byte
    - 0xff
    - 0x0e10
    - 0x07000000
- Compiler always creates "complex" literal instead of burning 2-3 insns to generate it
  - i.e. 0x07000400

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Bus Review

Table 2-1 : Memory Configuration and Specifications

Memory Type	Bus Width	Access Cycle	Bit Width that Allows DMA Access		Bit Width that Allows Main Processor Access	
			Read	Write	Read	Write
DS Accessory RAM (SRAM, flash memory, etc.)	8	6-18	-	-	8	8
DS Accessory ROM (ROM, flash memory, etc.)	16	1st 6-18 2nd 4-6	16/32	16/32	8/16/32	16/32
OAM	32	1	16/32	16/32	8/16/32	16/32
VRAM	16	1	16/32	16/32	8/16/32	16/32
Palette RAM	16	1	16/32	16/32	8/16/32	16/32
I/O Registers	32	1	16/32	16/32	8/16/32	8/16/32
Internal Work RAM	32	1	16/32	16/32	8/16/32	8/16/32
Main Memory	16	1st R:5 / W:4 2nd 1	16/32	16/32	8/16/32	8/16/32
System ROM	32	1	-	-	8/16/32	-
TCM/Cache	32	½	-	-	8/16/32	8/16/32

The values given for the number of access cycles correspond to a bus frequency of 33.514 MHz.

Furthermore, these values are for when memory is accessed in a bit width that is equal to or less than the bus width. When memory is accessed in a bit width that is larger than the bus width, the number of access cycles is limited to the bit width divided by the bus width.

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Bus Review

- Most important data moves via 16-bit bus
  - Think about that for a second
- IO register space is 32-bit
  - Graphics chip settings, 3d commands
  - Sprite data (OAM) access
  - Game Card access
- DMA incurs same penalties as CPU
  - But benefits from "burst mode"

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Burst Mode of DMA gives it an advantage for bulk data transfers, and is described in the NITRO Programming Manual.

# ARM9 Bus Review

- Not shown: VRAM contention
  - If a bank is being used by the graphics engines and you try to access it with CPU, a stall occurs on ARM9
  - Reverse: Causes graphics to flicker
  - How much of a stall?
    - Core clock : dot clock == 6:1
    - ARM9 clock : dot clock == 12:1
    - Most 2d data access takes multiple dot clocks

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

Obviously, if a bank isn't mapped to ARM9 then there's no contention..

Bottom line: don't access VRAM / graphics registers outside of v-blank

# ARM9 Bus Review

## Bus Arbitration

- What happens when ARM9 and ARM7 try to access main memory at the same time?
  - ARM7 wins (due to EXMEM priority setting)
  - Once a client has the bus no one can interrupt them
- How long does a client lock the bus?
  - Duration of transaction!
  - So don't do huge transfers; you stall all other clients!

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Problems caused by large bus transactions are mostly noticeable when wireless is on or MIC sampling is turned on at a high frequency.

Odds are good that you'll see little problem with large transactions for many single player games. But your audio use can impact that...



# ARM9 Bus Review

## Bus Arbitration

- Client Priority:
  1. ARM7 DMA
  2. ARM7 CPU
  3. ARM9 DMA
  4. ARM9 CPU

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# ARM9 Cache review

- 8KB I-Cache
- 4KB D-Cache
  - Warning: Additional (active) threads eat up space due to stack activity
- 32byte cache lines
  - You can lock down a chunk of RAM
    - 1k increments, 32byte aligned
  - Tricky to use it correctly
    - Usage puts pressure on existing lines
    - You need to make sure that the code or data is linked with the correct alignments
- Line must be filled before execution can resume

DEVELOPERS  
CONFERENCE

Nintendo Confidential



As noted earlier, you can only pre-load the instruction cache.  
Also, the main thread stack is located in DTCM

# ARM9 TCM review

- Tightly Coupled Memory
  - 32KB ITCM
    - Can store code+data
  - 16KB DTCM
    - Data only; never seems to be enough of it
  - TCM access does not use the bus!
- TCM is not dynamically allocatable
  - Well, tools don't support it nicely
  - But you can build your own support anyways
    - And you should. It's not too hard to do.

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Main thread stack is allocated in DTCM by default

## A Quick Summary..

- Bus contention for Main Ram or VRAM can stall the ARM9, slowing down your game
- No need for the bus:
  - TCM access
  - In-cache code and data access
- Needs the bus:
  - Main Memory access
  - VRAM access
  - Some memory-mapped I/O registers

DEVELOPERS  
CONFERENCE

Nintendo Confidential



## ..and some numbers

- Bus speed: 33.514MHz
- ARM9 speed: 67.028MHz
- Bus Width ↔ Main Memory: 16bits
- ARM9 Cache line: 32 bytes
- ARM instructions: 32bits each
- THUMB instructions: 16bits each
- D-Cache: 1024 32-bit words *total*
  - Some of it lost to literal pool & code overlap
  - And stacks for additional threads

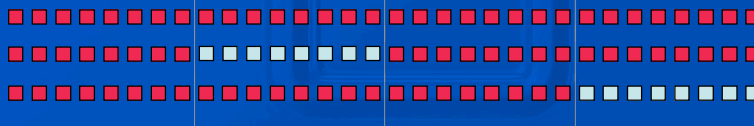
DEVELOPERS  
CONFERENCE

Nintendo Confidential

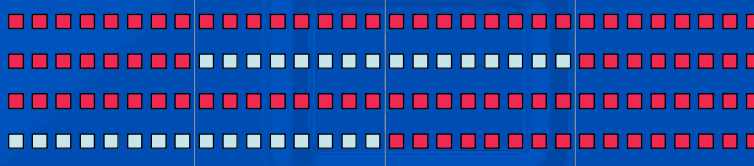


# Implications: I-Cache miss

32bit ARM Code:



16bit THUMB Code:



Stalled Running

Nintendo Confidential

Nintendo

DEVELOPERS  
CONFERENCE

- ARM9 goes 40+ cycles in the time that it takes the bus to fill an entire cache line (ie, 20 bus cycles)
  - Then gets to execute (AT MOST) 8 or 16 instructions before needing another cache line
- If that cache line isn't available, then the process repeats

# Run the numbers

- On I-Cache miss:
  - ARM: 40+8 cycles of time for 8 instructions
  - THUMB: 40+16 cycles of time for 16 instructions
- D-Cache miss is similar
  - 40 cycles later, you get your data
  - Hope you wanted more than 1 word from the cache line

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

ARM:

- $8/48 = 1/6 = 16\%$  efficiency
- 80 instructions could take up to 480 cycles when i-cache always misses

THUMB:

- $16/56 = 2/7 = 28\%$  efficiency
- 80 instructions could take up to 285 cycles when i-cache always misses

One could argue that ARM code is 1.5x (or more) efficient than THUMB, but THUMB makes up for it by virtue of cache wins. Might end up being a tie, but we're thinking that THUMB is a slight win overall.

# Implications

- Your game is memory bound
- This is why Cache and TCM are so incredibly important
  - And why all the interrupt handlers are in TCM
- Locality is key to performance

DEVELOPERS  
CONFERENCE

Nintendo Confidential





# Don't Panic!

250+ games on DS and no one has really noticed. Consumers don't care. So this only impacts you if you are *really* pushing the hardware.

DEVELOPERS  
CONFERENCE

Nintendo Confidential





Also might impact you if you're  
inadvertently running at < 20Hz

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Rules of THUMB

- Use 16bit THUMB instructions
  - As fast as 32bit instructions, but ½ the size
    - Crams more code into the Instruction Cache
  - Switching ARM↔THUMB is free ("blx" insn)
  - May cause literal pool to be slightly larger
  - But great if you are tight on RAM
    - And who isn't?
  - Easy to do:
    - `#include <nitro/code_16.h>`

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Rules of THUMB

- Functions that aren't called often ought to be in THUMB mode anyway
  - Initialization code is usually large. THUMB it!
- Functions that are simple should be in THUMB mode too
  - Most "getter" functions are suitable
  - Simple Boolean tests, too
- Disassemble your code in both ARM and THUMB and choose the version that gives best size/cycle tradeoff

DEVELOPERS  
CONFERENCE

Nintendo Confidential



If there is more than one bit operation (mask, shift, insert, extract) then ARM code is usually the winner.

But for cases where simple loads, stores, or comparisons are done then THUMB is usually 0 to 3 instructions larger and is half the size of ARM code.

# Rules of THUMB

- Remember when I said that THUMB instructions were just as fast as ARM instructions?
  - Not exactly true.. there's interlock involved with most of them which causes some stalls on each cycle
  - But you're totally stalled by the bus most of the time anyways so it doesn't really matter
  - And you get 2x THUMB code per cache line, so realistically you're getting more work done
- SDK defaults to 100% ARM code (for *both* processors) unless you specify otherwise

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Rules of THUMB

Some downsides

- Fewer registers available means more traffic to the stack
  - Most instructions can only access r0-r7
  - Limited instructions for access of r8-r15
- No win for branch-heavy calls
  - Jumping to other functions usually take 2 16-bit instructions back-to-back
    - so 32bits per branch - not a win

DEVELOPERS  
CONFERENCE

Nintendo Confidential



And now for  
something different



DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Main Memory Display Mode

- Possible Use:
  - Can use it while 2d/3d is being captured to VRAM
  - Generate a data on the fly or show static screen
  - Manually post-process a captured image
- Notes:
  - Once the mode is active, you MUST keep feeding it new data
  - Otherwise, it uses last data in the FIFO (ugly!)
  - Takes more hand holding than VRAM display mode

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Anybody using this? Would love some feedback on that.

Why use it?

1. Frees up VRAM for other purposes
2. You can be drawing the screen just ahead of the DMA read stream

Why not use it?

1. Can't exec auto-start DMAs



# Main Memory Display Mode

## DMA

- Uses AutoStart signal from LCDC
  - Data copy happens when the pixel display FIFO has room for another 4 words
  - Transfers 4 u32's, then goes idle (off bus)
  - Still Enabled, though, so channel is "locked"
- DMA completes when you hit V-Blank
  - You'll need to re-start the DMA for each upcoming frame
  - Do re-start during V-Blank
- Can't use other auto-start DMA modes
  - Immediate mode DMA ok, though

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

It's not necessary to pre-draw the whole screen before starting the transfer, but make sure not to let it grab garbage.

Roughly: Transfer activity happens around time that line is being drawn to LCD; goes idle during H-Blank

# Card DMA

- Card DMA functions similarly to Main Memory Display DMA
  - DMA channel is enabled for duration of entire transfer
  - Card AutoStart signal tells it to do some work
    - Only one u32 is transferred at a time!
    - Then goes idle (off the bus)
  - The cycle repeats until transfer is complete
    - Once all data is transferred, DMA is set to Disabled

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Remember: If the SDK usually checks a DMA channel before attempting to use it.

If the channel is marked as "Enabled", the SDK will stall until it goes to "Disabled".

So bad things happen if (for example) your FS and GX are set to use the same DMA channel.

•Technically, the auto-start only happens on a per-card-page basis. Our SDK sorta hides this from you to make life easier.

## Rules for Async Card read

- Follow the CARD\_ReadRomAsync guidelines
  - Make sure that all data is aligned to 512byte boundaries on the card (.rsf can specify this)
  - Read multiples of 512bytes
  - Target destination must be 32byte aligned
- Use DMA channel 3
  - Use higher channel numbers for GX & WM
- Cache Notes

DEVELOPERS  
CONFERENCE

Nintendo Confidential



Async is best if you can pre-load data before you actually need to display it.

Pre-load your splash screens while the first one is being displayed.

Pre-load your menu while the final splash screen is being displayed.

Take a best-guess at what level the player will be playing, and pre-load as much as you can *before* the user requests to move from your menus into the game.

This is the only way to get that GBA-like quality of "instantaneous" level loads

Cache notes: upcoming patch to SDK 4.x will make large data loads happen faster, due to invalidating the entire cache rather than one line at a time.

# Interrupt processing

- Get in, then get out
  - If possible, just set some flags and deal with it later
  - Sometimes you *do* have to do actual work, but make it quick
- This is an embedded device
  - Not a lot of time to burn; cycles are precious
  - Try to put the callbacks in ITCM
- You will destabilize the system if you take too long in a callback or interrupt handler!
  - SDSG has seen WAY too many cases of “heavy lifting” taking place within an interrupt handler
  - And we’ve seen the chaos it causes.
  - We thought you ought to know...

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# V-Blank Handler

60Hz (during production)

```
#include <nitro/itcm_begin.h>

NITRODEVCAPS _devCaps;

static void handle_vbl_intr(void)
{
    if (
        _devCaps.m_dwMaskResource & NITROMASK_RESOURCE_VBLANK
    )
        NITROToolAPIVBlankInterrupt();

    // set the flag saying that we've
    // dealt with the interrupt.
    OS_SetIrqCheckFlag(OS_IE_V_BLANK);
}

#include <nitro/itcm_end.h>
```

Nintendo Confidential



# V-Blank Handler

60Hz (finalrom)

```
#include <nitro/itcm_begin.h>

static void handle_vbl_intr(void)
{
    // set the flag saying that we've
    // dealt with the interrupt.
    OS_SetIrqCheckFlag(OS_IE_V_BLANK);
}

#include <nitro/itcm_end.h>
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential





Speaking of V-Blank...

DEVELOPERS  
CONFERENCE

Nintendo Confidential



## Fast data upload for V-Blank

- Don't wait until V-Blank to decide what data to upload
- Determine all necessary info ahead of time
  - Function that you'll use to do the upload
  - Source and Destination Addresses
  - Byte count to xfer, VRAM offset
- Put this info in a list or queue
- Then run through the list when V-Blank hits and dispatch it all

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

Actually, you might want one list for 3d items, and then another list for everything else. This is because the 3d V-Blank window is quite small, and happens at the beginning of the V-Blank period.



# Why?

- Will simplify game logic
- Will simplify V-Blank processing
- Makes the most of V-Blank time

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Fast data upload for V-Blank

```
// can only call during V-Blank  
GX_LoadOBJPltt(pObjPltt, 0, 32);
```

→ // want ability to call this anytime during the frame

```
defer(GX_LoadOBJPltt, pObjPltt, 0, 32);
```

→ // so main loop looks something like..

```
while(true)  
{  
    // ...  
    OS_WaitVBlankIntr();  
    dispatch_deferred_functions_then_clear();  
    // ...  
}
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Fast data upload for V-Blank

This effectively gets us:

```
while(true)
{
    // ...
    OS_WaitVBlankIntr();
    GX_LoadOBJPltt(pObjPltt, 0, 32); // was deferred
    // ...
}
```

not actual code

- But you can dynamically have many different calls
- And they can vary on each frame
- Gives us immense flexibility

DEVELOPERS  
CONFERENCE

Nintendo Confidential



## Notes on the Deferred Function Handler

- Can call any number of functions
  - Limited by RAM
  - FIFO call order
- Each call can have between 0-4 params
- System provides temporary storage space in case you need to record a value in time

```
// 3 params  
GX_LoadOBJPltt(pObjPltt, 0, 32);
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Fast data upload for V-Blank (code)

```
asm void dispatch_deferred_functions_then_clear(register dfpPtr list)
{
    stmfd    spl, {r4-r10, pc}
    ldr      curPtr, [list, #8]           // get start of list
    ldr      lastPtr, [list, #0]         // get end of list

    cmp      curPtr, lastPtr             // compare curPtr-lastPtr
    beq      finish                     // branch if curPtr == lastPtr
    mov      dfpBlockPtr, list           // stash away list pointer for later

loop:
    // 1) load packed control word + functionAddr
    // 2) then extract num args + bytes to skip from control word
    ldmba    curPtr, [bytesToSkip, procAddr] // load ctrlWord + procAddr
    mov      argsToLoad, bytesToSkip, LSL #28 // lower 4 bits are num args to load
    mov      bytesToSkip, bytesToSkip, LSR #4 // remaining bits are bytes to skip

    // argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
    msr      CPSR_F, argsToLoad

    // now load the appropriate set of registers based on the condition bits. order is: n,z,c,v
    ldmbvia  curPtr!, {r0}               // <cond>=vs == v flag only
    ldmbvia  curPtr!, {r0-r1}            // <cond>=cs == c flag only
    ldmbvia  curPtr!, {r0-r2}            // <cond>=eq == z flag only
    ldmbvia  curPtr!, {r0-r3}            // <cond>=ml == n flag only

    add      curPtr, curPtr, bytesToSkip // add any left over bytes
    blx      procAddr                    // make the call
    cmp      curPtr, lastPtr             // set flags to result of: (curPtr - lastPtr)
    bni      loop                        // branch if (curPtr < lastPtr) ; ie, we haven't reached the end

finish:
    // now set curPtr back to bufferStart
    ldr      r0, [dfpBlockPtr, #8]       // start of list
    str      r0, [dfpBlockPtr, #0]       // cur now == start of list
    ldmbfd   spl, {r4-r10, pc}           // load multiple (frame descending) w/ update
}
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo



# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}       // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}       // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}       // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}       // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}       // <cond>eq == z flag only
ldmmiaa    curPtr!, {r0-r3}       // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                    // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmiaa    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}       // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}       // <cond>eq == z flag only
ldmmiaa    curPtr!, {r0-r3}       // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                    // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}        // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}        // <cond>eq == z flag only
ldmmlia    curPtr!, {r0-r3}        // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                     // make the call
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

# Fast data upload for V-Blank

(code)

```
// 1) load packed control word + functionAddr
// 2) then extract num args + bytes to skip from control word
ldmia      curPtr!, {bytesToSkip,procAddr}
mov  argsToLoad, bytesToSkip, LSL #28
mov  bytesToSkip, bytesToSkip, LSR #4
// argsToLoad got converted to condition bits.. move em into CPSR_FLAGS
msr  CPSR_F, argsToLoad
// now load the appropriate set of registers based on the condition bits.
// order is: n,z,c,v
ldmvsia    curPtr!, {r0}           // <cond>vs == v flag only
ldmcsia    curPtr!, {r0-r1}       // <cond>cs == c flag only
ldmeqia    curPtr!, {r0-r2}       // <cond>eq == z flag only
ldmmiia    curPtr!, {r0-r3}       // <cond>mi == n flag only
add  curPtr, curPtr, bytesToSkip // add any left over bytes
blx  procAddr                    // make the call
```


DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Fast data upload for V-Blank

(code)



```
void GX_LoadOBJPltt(const void *pSrc, u32 offset, u32 szByte)
{
    SDK_NULL_ASSERT(pSrc);
    SDK_ASSERT(offset + szByte <= HW_OBJ_PLTT_SIZE);
    SDK_ALIGN2_ASSERT(offset);
    SDK_ALIGN2_ASSERT(szByte);

    GXi_DmaCopy16(GXi_Dmald, pSrc, (void *) (HW_OBJ_PLTT + offset),
        szByte);
}
```

DEVELOPERS  
CONFERENCE

Nintendo Confidential





## Fast data upload via Deferred Function Calls

- No more if / switch logic at V-Blank
  - Or worse
- Maximize upload time
- "Fire and forget" graphics upload commands during game logic
- Flexible function call system
- Can be used for other things, too

DEVELOPERS  
CONFERENCE

Nintendo Confidential



## One small problem...

(which we've mentioned before)

- Might not want to upload a huge chunk of data via single DMA
  - because it blocks all clients on the device
- Problematic when Audio and Wireless are in heavy use
- Solution: Chunk up your data uploads into smaller DMAs so that the other clients get access to Main Memory

DEVELOPERS  
CONFERENCE

Nintendo Confidential

Nintendo

Experiment to see what works best. Start with 1KB chunks and scale up or down from there.

# Async functions

- Not all functions are equally asynchronous
- Some dispatch work to ARM7, others to the DMA controller, and yet others queue work for separate threads
- Functions such as FS\_ReadFileAsync will actually fall into synchronous mode if the parameters aren't "perfect"

# Recap

- ARM7 CPU operation
- ARM9 CPU operation
  - And interaction with main memory
- Bus operation
- DMA controllers
- V-Blank handling
- And More!

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Take-away

- Come away with a better, deeper understanding of overall system design
  - And how to fit into it
- Knowledge to build engine and game code to make best use of Nintendo DS

DEVELOPERS  
CONFERENCE

Nintendo Confidential



# Advanced Programming Topics for Nintendo DS

Mark Jawad

Senior Software Engineer

Software Development Support Group

DEVELOPERS  
CONFERENCE

Nintendo Confidential

