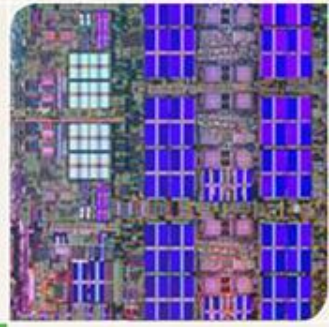
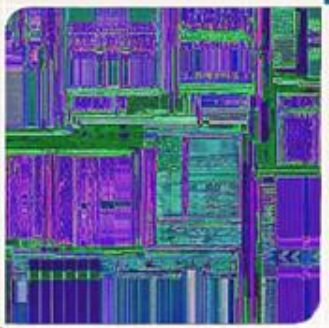
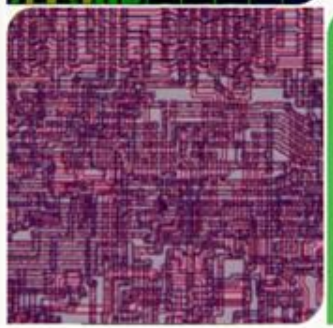



CS4803DGC Design and Programming of Game Console

Spring 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing

Nintendo DS/DSi/ DSlite

Georgia Tech College of Computing



Introduction

- 2nd part of programming platform
- Programming with Nintendo DS
- http://www.cc.gatech.edu/~hyesoon/spr11/lab4.html
- Installation Guide and Hello world

Hardware Picture

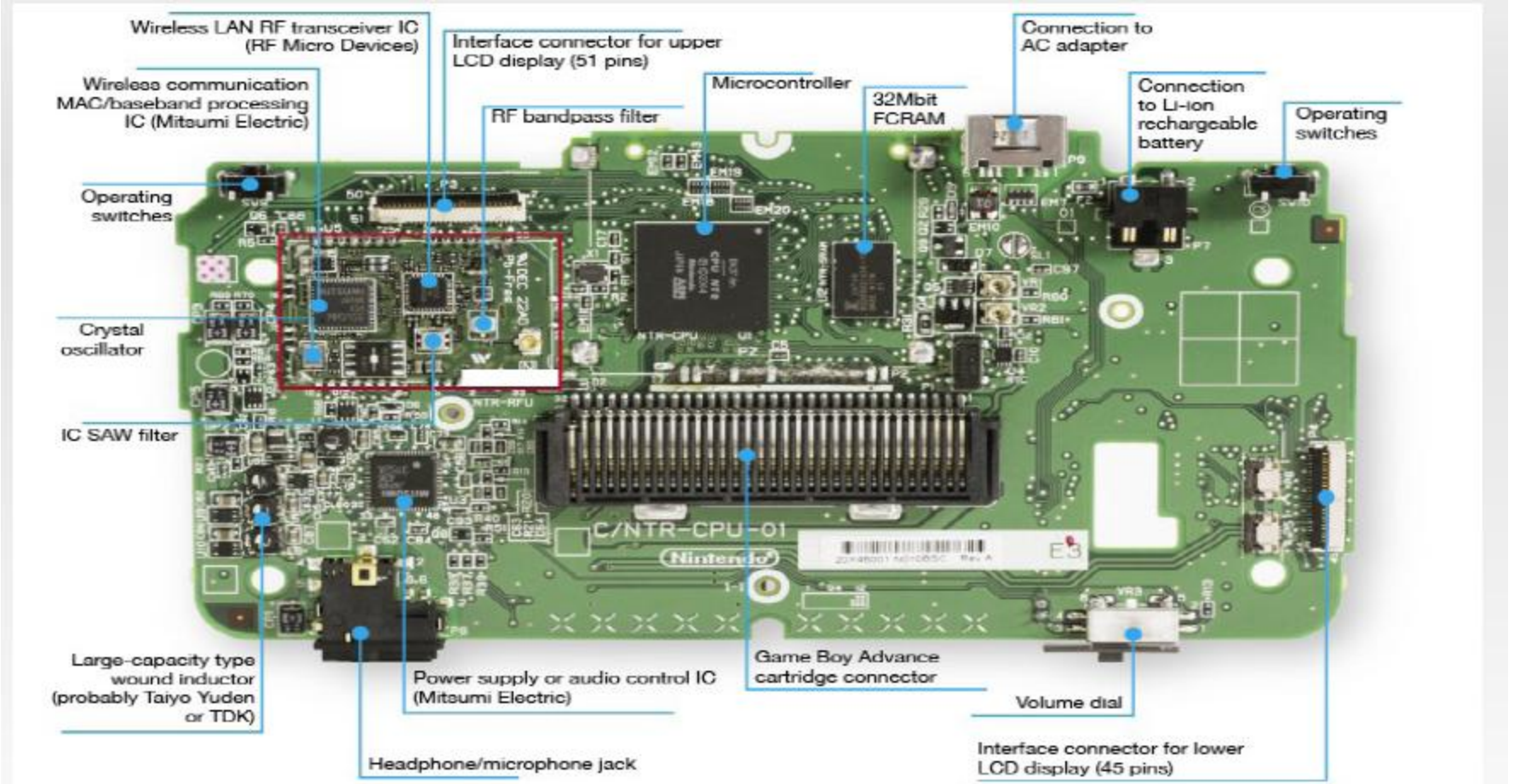


Fig 1 Key Circuitry Clumped on One Side The photo is close to the actual size – 141 x 76mm (longest area). The wireless module and microcontroller were covered by a metal shielding plate.

http://www.cosc.brocku.ca/Offerings/3P92/seminars/nintendo_ds_slideshow.pdf



Hardware Specifications

- Dual TFT LCD screens
- CPUs
  - ARM 7 TDMI (33MHz)
  - ARM 9 946E-S (67MHz)
- Main memory: 4MB RAM
  - VRAM: 656 KB
- 2D graphics
  - Up to 4 backgrounds
- 3D graphics



ARM7/ARM9

- Both can be running code at the same time.
- ARM 7 is the only CPU that controls the touch screen.
  - Interrupt based



DevKit Pro & libnds

- DevKit Pro is a collection of tool chain for homebrew applications developers for various architectures
- DevKitARM: ARM binaries
- Not official development tool chain
  - Much simpler and naïve
- libnds
  - Started with header files for definition
  - Extended to have other data structures, simple APIs
- *.nds
  - A binary for Nintendo DS, a separate region for ARM7 and ARM9

http://patater.com/files/projects/manual/manual.html#id2612503



Review Hello World

```
int main(void) {

consoleDemoInit(); //Initialize the console

irqSet(IRQ_VBLANK, Vblank); //this line says: When the IRQ_VBLANK
    interrupt occurs execute function Vblank
iprintf("      Hello DS dev'rs\n");

while(1) {
    iprintf("\x1b[10;0HFrame = %d",frame); //print out the current frame number
    swiWaitForVBlank(); //This line basically pauses the while loop and makes it
    //wait for the IRQ_VBLANK interrupt to occur. This way, we print only once
    //per frame.
    }
    return 0;
    }
```



Assignment #4

- ARM assembly code
  - Up: OR operation
    Down: AND operation
    start: Reset to default values
    A: Exclusive OR operation
    B: AND NOT (BIC) operation
    Left: left shift by #1
    Right: right shift by #1
    No need to use interrupt, use a polling method

GCC Inline Assembly Programming

- Instead of pure assembly coding, we will use inline assembly programming
- Not only ARM, x86 etc.
- Good place to look at

http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html#ss5.3

http://www.ethernut.de/en/documents/arm-inline-asm.html

NOP

```
asm( "mov r0, r0\n\t"
     "mov r0, r0\n\t"
      "mov r0, r0\n\t"
       "mov r0, r0" );
```

Use deliminaters Linefeed or tab to differentitate assembly lines



ASM Examples

```
asm(code :  // opcode , destination, src
        output operand list : /* optional*/
        input operand list : /* optional*/
        clobber list /* optional*/
);
/* Rotating bits example */
  asm("mov %[result], %[value], ror #1" :
            [result] "=r" (y) :
            [value] "r" (x));
```

Symbolic name encoded in square brackets
followed by a constraint string, followed by a C expression enclosed in parentheses
e.g.) sets the current program status register of the ARM CPU

```
asm("msr cpsr,%[ps]" :
                              :
            [ps]"r"(status)
  );
```



Constraint of Input and Output Operands

Constraint	Usage in ARM state	Usage in Thumb state
f	Floating point registers f0 .. f7	Not available
h	Not available	Registers r8..r15
G	Immediate floating point constant	Not available
H	Same a G, but negated	Not available
I	Immediate value in data processing instructions e.g. ORR R0, R0, #operand	Constant in the range 0 .. 255 e.g. SWI operand
J	Indexing constants -4095 .. 4095 e.g. LDR R1, [PC, #operand]	Constant in the range -255 .. -1 e.g. SUB R0, R0, #operand
K	Same as I, but inverted	Same as I, but shifted
L	Same as I, but negated	Constant in the range -7 .. 7 e.g. SUB R0, R1, #operand
l	Same as r	Registers r0..r7 e.g. PUSH operand
M	Constant in the range of 0 .. 32 or a power of 2 e.g. MOV R2, R1, ROR #operand	Constant that is a multiple of 4 in the range of 0 .. 1020 e.g. ADD R0, SP, #operand
m	Any valid memory address	
N	Not available	Constant in the range of 0 .. 31 e.g. LSL R0, R1, #operand
O	Not available	Constant that is a multiple of 4 in the range of -508 .. 508 e.g. ADD SP, #operand
r	General register r0 .. r15 e.g. SUB operand1, operand2, operand3	Not available
w	Vector floating point registers s0 .. s31	Not available
X	Any operand	

Modifier

Modifier	Specifies
=	Write-only operand, usually used for all output operands
+	Read-write operand, must be listed as an output operand
&	A register that should be used for output only

```
asm("mov %[value], %[value], ror #1" : [value] "+r" (y));
```

Same register value

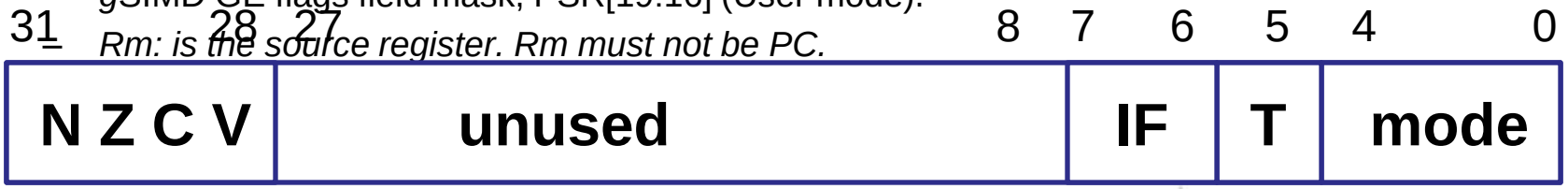
Assembly Instruction

- MOV
  - MOV{S}{cond} Rd, Operand2
  - MOV{cond} Rd, #imm16
- MSR

  Load an immediate value, or the contents of a general-purpose register, into specified fields of a *Program Status Register* (PSR)

  Syntax

  MSR{*cond*} APSR_*flags*, *Rm* where:
  - *Cond* is an optional condition code.
  - *Flags* specifies the APSR flags to be moved. *flags* can be one or more of:
  - *Nzcvq* ALU flags field mask, PSR[31:27] (User mode)
  - *g*SIMD GE flags field mask, PSR[19:16] (User mode).
  - *Rm: is the source register. Rm must not be PC.*





Clobber List

- Some instructions clobber some hardware registers.
- We have to list those registers in the clobber-list

Example: Simple ADD and Print

```
nt main(void) {
//---------------------------------------------------------------------------
        consoleDemoInit();
        int* notGood= (int *)0xb0; //bad
        *notGood= 10;
        int better=20;
        irqSet(IRQ_VBLANK, Vblank);
        printf("      Hello CS4803DGC");
// case 1
        asm("MOV R1,  #0xb0"); //init R1 to address
        asm("LDR R0, [R1]");
        asm("ADD R0, R0, R0");
        asm("STR R0, [R1]");
// case 2
        asm ("MOV R1, %[value]"::[value]"r"(better));
        asm ("ADD R1, R1, R1");
        asm ("MOV %[result], R1":[result]"=r"(better):);
        while(1) {
                swiWaitForVBlank();
                // print at using ansi escape sequence \x1b[line;columnH
                printf("\x1b[10;0HFrame = %d",frame);
                printf ("\nblah is: %d, %d", *notGood, better);
}
        return 0;
}
```

Please note that this code does not run correctly!

Nintendo DS Input System

- Button, touch screen, microphone
- Libnds key definition

KEY_A	1 << 0	A Button
KEY_B	1 << 1	B Button
KEY_SELECT	1 << 2	Select Button
KEY_START	1 << 3	Start Button
KEY_RIGHT	1 << 4	Right D-pad
KEY_LEFT	1 << 5	Left D-pad
KEY_UP	1 << 6	Up D-pad
KEY_DOWN	1 << 7	Down D-pad
KEY_R	1 << 8	R Button
KEY_L	1 << 9	L Button
KEY_X	1 << 10	X Button
KEY_Y	1 << 11	Y Button
KEY_TOUCH	1 << 12	Pen Touching Screen (no coordinates)
KEY_LID	1 << 13	Lid shutting (useful for sleeping)

Memory Mapped I/O

0x4000130	

Key Mapping in Nintendo DS

- The current status of the keys is stored in memory at address 0x4000130.
- When no key is pressed- the value is 1023.
- A key press causes a change in the value at this location. The new value depends on which key is pressed.
- Here are the values for various keys.

  A- #1022 b 11 1111 1110

  B- #1021 b 11 1111 1101

  start- #1015 b 11 1111 1011

  UP- #959 b 11 1011 1111

  DOWN- #895 b 11 0111 1111



Example: Reading Keys

```
asm ("MOV R4, #0x0000"); //R4 has the counter.. funny things
    happening with R1
      while(1) {
    swiWaitForVBlank();
       //init R4 to address
    asm ("MOV R0, #0x4000000"); //R0 has the address
    asm ("ADD R0, #0x130");  //        finished moving address
    /* We have only 8-bit immediate values */

//load value from that address
    asm ("LDR R2, [R0]");
// check the register value of R2 and compare and then increment the
    counter
    // use condition code or shift etc.
  //move counter value from R2 to C variable
    asm ("MOV %[result], R2":[result]"=r"(result_):);
```



Caution!

- Compiler still rearranges the assembly code.
- Default compilation mode is ARM-thumb
- The makefile has to be modified- set it to no optimization by -O0

- change line ARCH := -mthumb –mthumb-interwork TO ARCH := -marm