# EECS 373 *Midterm*

Winter 2014

Name: ____________________________________ unique name: _______________

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so. <u>Nor did I discuss this exam with anyone after it was given to the rest of the class.</u>

___________________________________

Scores:

| # Page | Points |
|---|---|
| 2 | /12 |
| 3 | /10 |
| 4 | /15 |
| 6 | /18 |
| 9 | /30 |
| 10 | /15 |
| **Total** | **/100** |

## NOTES:

1. Closed book/notes and no calculators.
2. There are 11 pages including this one. ***The last page is a reference sheet. You may wish to rip it out.***
3. Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
4. Don't spend too much time on any one problem. If you get stuck, move on!!!
5. You have about 80 minutes for the exam.
6. **Be sure to show work and explain what you've done when asked to do so.** Getting partial credit without showing work will be rare.
7. **Throughout the exam "standard logic gates" means arbitrary input ANDs, ORs, NANDs, NORs, XORs and XNORs as well as NOT gates.**

1) Multiple-choice/fill-in-the-blank **[12 points, -2 per wrong or blank answer, minimum 0]**

a. In Verilog an @* block is used to create ***combinational logic / sequential logic / a flip-flop***.

Combinational logic

b. 100 clock ticks of a 100MHz clock takes about __________ ns.

1000

c. A ____________ is when a combinational signal should stay at a level (0 or 1), but instead changes to the other level for a short time.

Bounce

d. When using an open collector scheme to drive wires on a bus means that if there are more than one devices trying to drive the bus in different ways the result will be a ***logical 1 / logical 0 / HiZ / creating a short circuit and letting out the "magic smoke"***.

Logical 0

2) Short answer. You should answer each question in three sentences or less. **[10 points]**

   a. Explain why 0x5555 and 0x55550000 can't both be instructions on the Cortex M3 **[5]**

      0x55550000 is a 32-bit instruction, and the M3 supports Thumb instructions which are 16-bits.

3) Say we have three potential interrupt sources, external exception numbers 50, 51 and 52. 52 is to be capable of preempting the other two interrupts, and 51 should have a higher priority than 50, but not be able to preempt it. ***<u>The page following this one has a number of tables and other information that you may find useful when solving this problem.</u>*** **[15 points]**

   a. Assuming PRIGROUP=6, provide priority levels for the three sources that would give the desired preemption and priority between them. Provide your answers as 8-bit numbers in hex. **[3]**

      Priority level for 50: ________0x81______ Priority level for 51: ____0x80_________

      Priority level for 52: ________0x00______

   b. Say that the interrupts and their ISRs had the following properties

| **Exception #** | **Time the interrupt occurs at** | **ISR run time** |
|---|---|---|
| 50 | 0ms | 5ms |
| 51 | 3ms | 3ms |
| 52 | 4ms | 2ms |

   At what time will the ISR for exception 50 finish running? *<u>Clearly justify your answer.</u>* **[6]**

Exception 50 will run from times 0-4, at which point it will be preempted by Exception 52. Exception 52 runs from times 4-6 at when it completes. Exception 51 then runs from time 6-9. Exception 50 will then complete by time 10.

Note that Exception 51 does not preempt Exception 50, although it is given higher priority after Exception 52 completes.

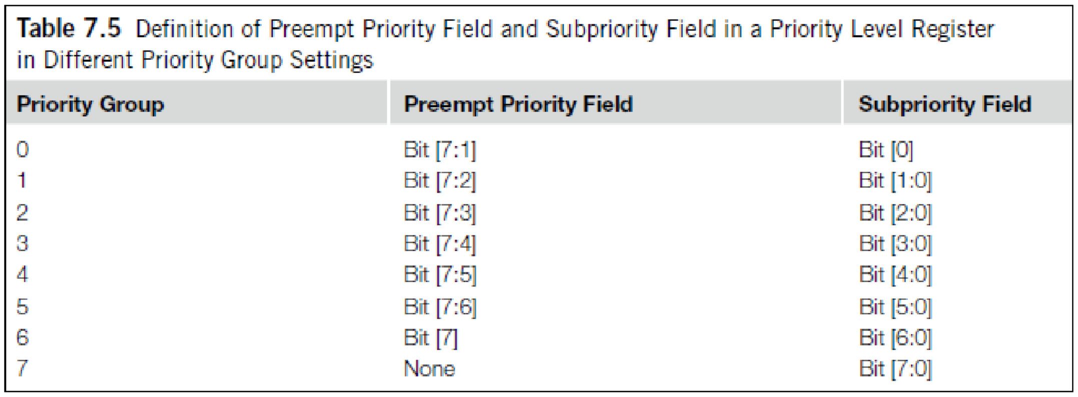


**Table 7.5** Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

| Priority Group | Preempt Priority Field | Subpriority Field |
|---|---|---|
| 0 | Bit [7:1] | Bit [0] |
| 1 | Bit [7:2] | Bit [1:0] |
| 2 | Bit [7:3] | Bit [2:0] |
| 3 | Bit [7:4] | Bit [3:0] |
| 4 | Bit [7:5] | Bit [4:0] |
| 5 | Bit [7:6] | Bit [5:0] |
| 6 | Bit [7] | Bit [6:0] |
| 7 | None | Bit [7:0] |

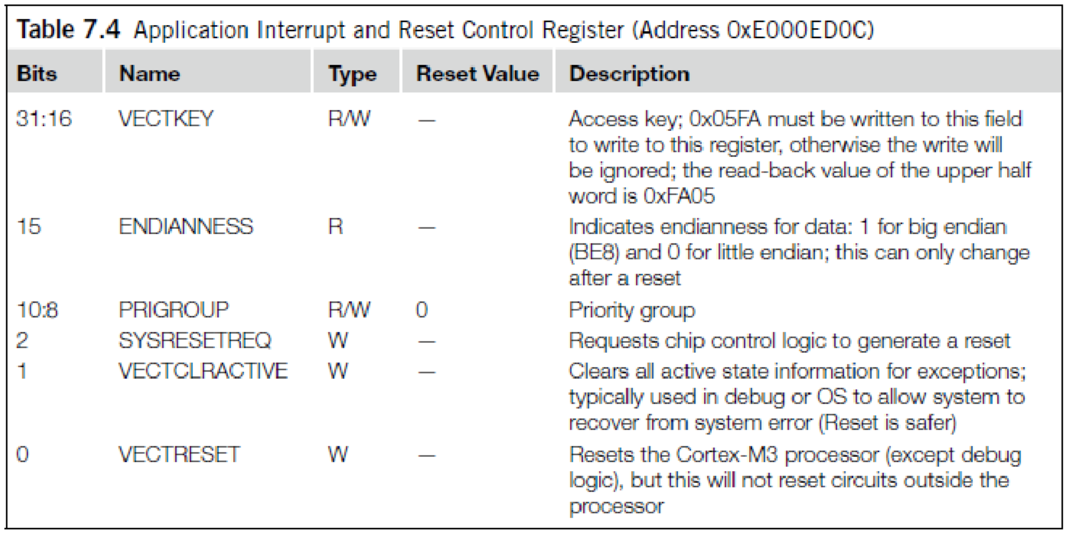
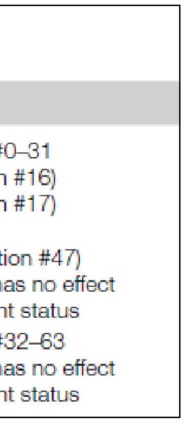


**Table 7.4** Application Interrupt and Reset Control Register (Address 0xE000ED0C)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 31:16 | VECTKEY | R/W | — | Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05 |
| 15 | ENDIANNESS | R | — | Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset |
| 10:8 | PRIGROUP | R/W | 0 | Priority group |
| 2 | SYSRESETREQ | W | — | Requests chip control logic to generate a reset |
| 1 | VECTCLRACTIVE | W | — | Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer) |
| 0 | VECTRESET | W | — | Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor |


#0–31
n #16)
n #17)
tion #47)
has no effect
nt status
#32–63
has no effect
nt status

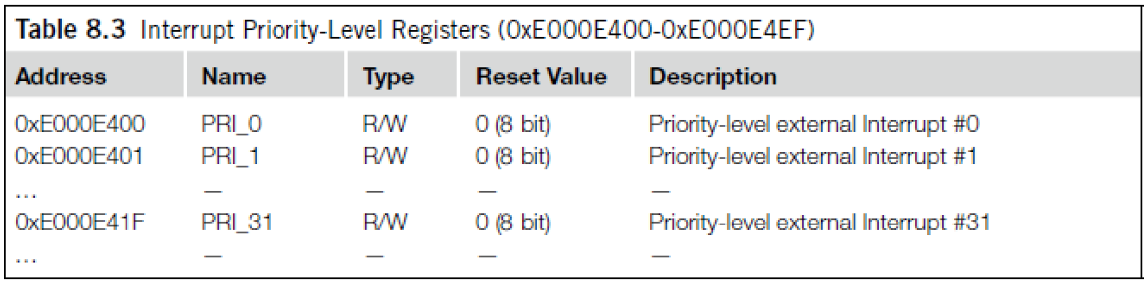


**Table 8.3** Interrupt Priority-Level Registers (0xE000E400-0xE000E4EF)

| Address | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 0xE000E400 | PRI_0 | R/W | 0 (8 bit) | Priority-level external Interrupt #0 |
| 0xE000E401 | PRI_1 | R/W | 0 (8 bit) | Priority-level external Interrupt #1 |
| … | — | — | — | — |
| 0xE000E41F | PRI_31 | R/W | 0 (8 bit) | Priority-level external Interrupt #31 |
| … | — | — | — | — |

4) Write an ARM assembly language procedure that implements the following C function in an EABI-compliant manner and conforms to the following signature. *Clearly comment your code* so we can figure out what we are doing and what value each register holds. Poorly commented/unclear code will get points removed. You are to assume “printit()” is an ABI-compliant function that already exists. **[18 points]**

```
uint32 Thing1(uint32 erf, uint32 x[])
{
        int y;

        printit(x[erf]);

        if(x[erf]>erf)
                y=x[erf];
        else
                y=x[1];
        return(y+1);
}
```

```
Thing1:
        mov r2, r0                  @r2 = erf
        ldr r0, [r1, r2, lsl #2]    @r0 = x[erf] must shift left to account for 32-bit int
        push{r0, r1, lr}            @ store scratch registers we want to save
        bl printit
        pop{r0, r1, lr}             @restore scratch regs
        cmp r0, r2
        ble else
if:
        mov r3, r0
        b endif
else:
        ldr r3, [r1,#4]
endif:
        mov r0, r3
        add r0, #1
        bx lr
```

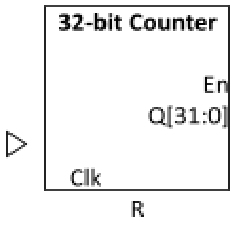
# Vibration Sensing Design Problem

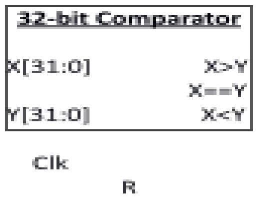
Engine knock occurs in a car engine when the piston fires before it should causing abnormal vibrations. Engine controllers monitor these vibrations and adjust the engine timing to prevent this. Your job is to monitor the vibrations and provide an alert to the engine management software when the vibrations become excessive.

Excessive vibration is determined by simply measuring the number of vibrations over some time interval. We don't want to use processor time to do this, so you will accumulate the counts with a simple 32-bit counter. *Each low-to-high transition of the vibration sensor counts as a single vibration.* The counter will be reset on regular basis (called the "sampling interval"). If the counter exceeds some fixed value (called the "count threshold"), an interrupt will be generated.

The sampling interval and count threshold are both to be writeable, while the current value of the counter tracking the number of vibrations during the sampling interval is to be readable. The hardware will specify both the sampling interval and the count threshold in terms of clock ticks of PCLK ticks. PCLK is 100MHz.

## Hardware available:





The following hardware devices are available (you may use as many of each as you need).

A 32-bit modulo[1] counter with enable and reset. Its inputs are clock and reset, while its output is simply the current count as a 32-bit number.

A 32-bit comparator that has two 32-bit inputs (X and Y) and three outputs (X>Y, X==Y, X<Y).

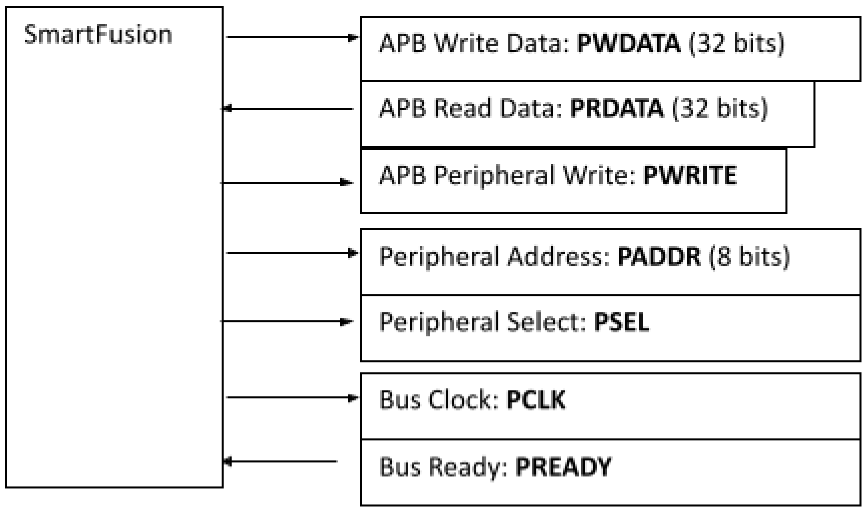
Standard gates, flip-flops, MUXes, and registers (with enable and reset as desired).

*You will likely find it helpful to read the rest of the exam before solving any of the following parts.*

[1] Recall that a "modulo" counter simply wraps around when it reaches its maximum value. So a 2-bit modulo counter would count 0,1,2,3,0,1,2,3,0, etc.

## Interface to the APB bus

The kit has the following APB3 bus interface. The signal names are shown in bold. The ABP3 bus signals follow APB3 timing and protocol. Read and write cycles are provided on the next page. PSEL is configured to be "1" when memory locations **0x40050000-0x4005000F** are accessed.


SmartFusion
APB Write Data: **PWDATA** (32 bits)
APB Read Data: **PRDATA** (32 bits)
APB Peripheral Write: **PWRITE**
Peripheral Address: **PADDR** (8 bits)
Peripheral Select: **PSEL**
Bus Clock: **PCLK**
Bus Ready: **PREADY**

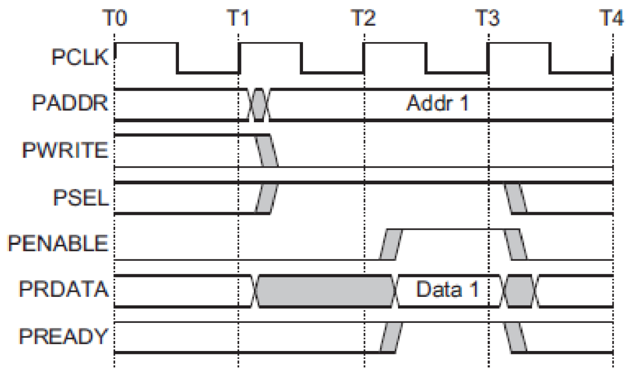


Figure 2-3 Read transfer with no wait states

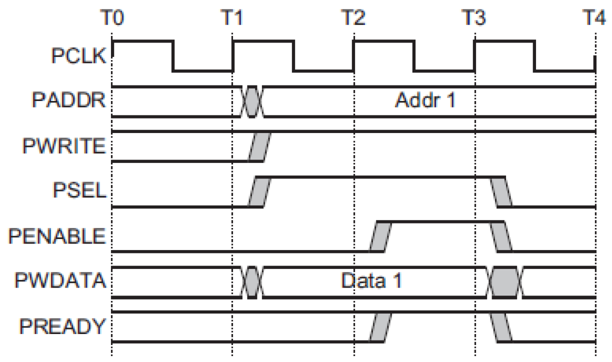


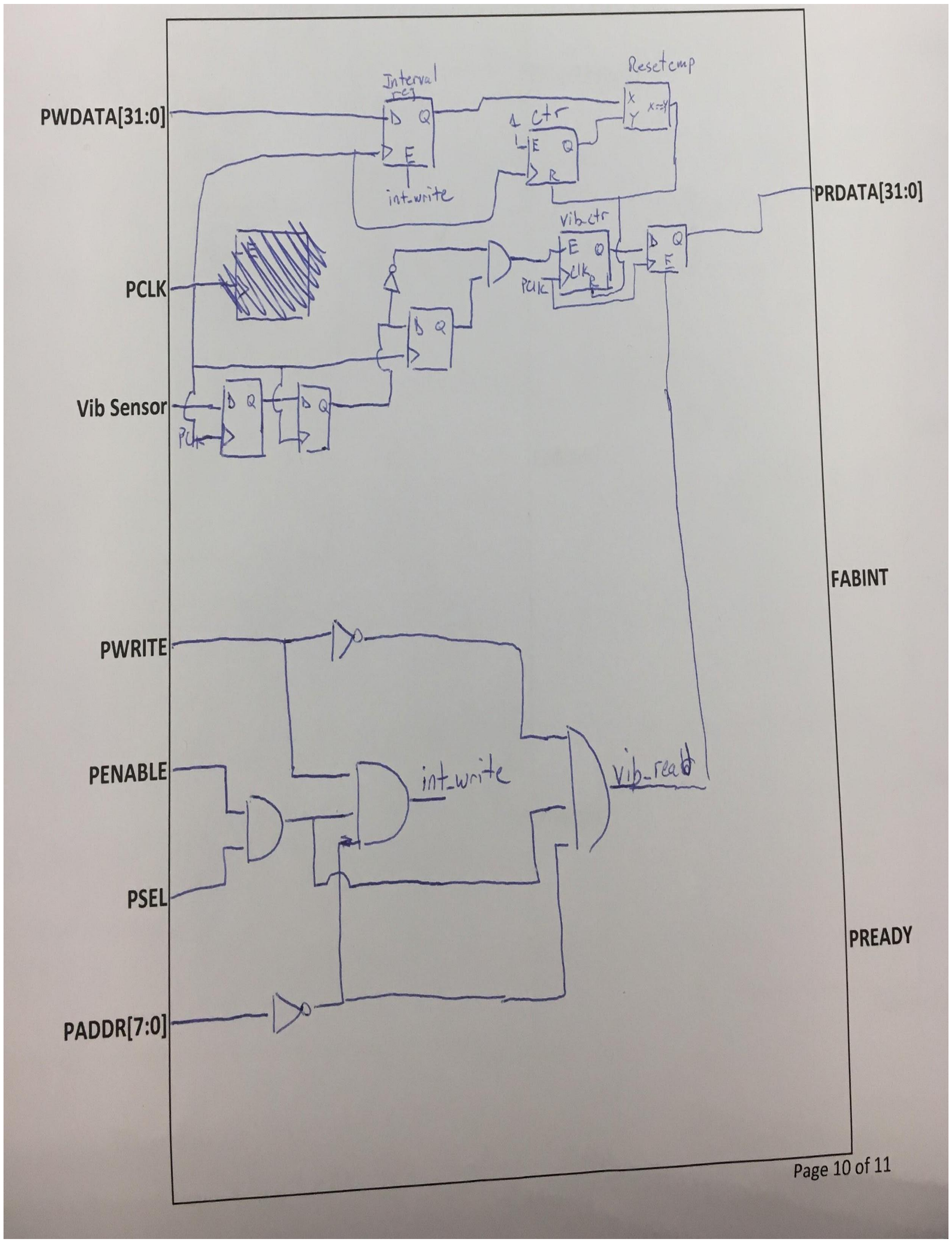
Figure 2-1 Write transfer with no wait states

## Part 1: Hardware **[30 points]**

Provide APB3 interfaces to the timer and counter such that.

A write to location **0x40050000** is to set the sampling interval to be the value written.
A write to location **0x40050004** is to set the count threshold to be the value written.
A read from location **0x40050000** returns the number of vibrations in this sampling interval.
An interrupt is driven for one clock tick with when the number of vibrations in the current sampling interval *exceeds* the count threshold.

Shadow locations are acceptable.



PWDATA[31:0]
PCLK
Vib Sensor
PWRITE
PENABLE
PSEL
PADDR[7:0]
PRDATA[31:0]
FABINT
PREADY
Interval reg
D Q
E
int_write
Resetcmp
X Y X==Y
ctr
E Q R
vib_ctr
E Q clk R
PCLK
D Q E
int_write
vib_read
Page 10 of 11