EECS 373 *Midterm*

Winter 2016

Name: _____ unique name: _____

Sign the honor code:

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

_____

NOTES:

1. Closed book/notes and no calculators.
2. There are 13 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. Nothing capable of wireless communication!
4. Don't spend too much time on any one problem. If you get stuck, move on!!!
5. You have about 120 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

    a. In Verilog an assign statement is generally used to create ***combinational logic / glitch-free logic / sequential logic / a flip-flop.***

    b. On the SmartFusion, priority groups ***1 and 2 / 2 and 4 / 4 and 5 / 6 and 7*** **are** all effectively the same in all cases (last page references may be useful here).

    c. If the priority group is set to be "7" then
        - ***interrupts are all enabled.***
        - ***no interrupt can preempt another interrupt.***
        - ***each interrupt source has a unique preemption priority.***
        - ***each of up to seven interrupt sources can have different priorities.***

    d. It is useful to have a "capture" register on a timer because:
        - ***it allows us to generate interrupts when an event occurs***
        - ***it allows us to generate an interrupt when a certain value is reached***
        - ***it can provide a more precise time of an event than reading the timer in software***
        - ***it provides a way to count the number of events which have occurred in a given amount of time.***

    e. A function pointer points to the
        - ***arguments passed to a function.***
        - ***part of the stack where the function should place data.***
        - ***start of the machine code of a function.***
        - ***location the function should return to.***

    f. One advantage of using an open-collector scheme over tri-state devices when using a "multi-drop bus" is that
        - ***It's easier to oversample the input.***
        - ***It's harder to "let out the magic smoke".***
        - ***It's easier to create a HiZ output.***
        - ***It's easier to drive really high currents.***

2. Answer the following questions **[5 points]**
    a. Certain devices, like a servo or an SPI-based distance sensor, can be readily controlled from a typical embedded processor. But others, like an N64 controller are much harder to control from a typical embedded processor and are better controlled using an FPGA.
    Explain why. **[5]**

3. Consider the following code. Assume the first mov instruction is at location 0x100 and that all *instructions* are 32-bits. **[8 points]**

```
data:
    .byte 0x10, 20, 0xf0, -2
func:
        mov r0, #0
        mov r4, #0
        movw    r1, #:lower16:data
        movt    r1, #:upper16:data
top:    ldrsb   r2, [r1],1
        add r4, r4, r2
        add r0, r0, #1
        cmp r0, #4
        bne top
done:   b done
```

- What is the value of the label "data" (in hex)? **[1]** _____
- What is the value of the label "top" (in hex)? **[1]** _____
- What is the final value of r0 (in decimal)? **[1]** _____
- What is the final value of r1 (in decimal)? **[2]** _____
- What is the final value of r4 (in decimal)? **[3]** _____

4. Write an EABI-compliant function in ARM UAL which implements the following C function “sum”. You can assume a short is 2 bytes in size and that “bob” is an ABI compliant function. **[10 points]**

```
int sum(short * list, int num)
{
   int i;
   int x=0, y;
   for(i=0;i<num;i++)
      x+=list[i];
   x=x-num;
   y=bob(list,x);
   return(x+y);
}
```

5. For the following program, assume you start with all memory locations in question equal to zero. Indicate the values found in these memory locations when the programs end. *Write all answers in hex.* Each memory location shown is a single byte. **[6 points, -1 per wrong box min 0]**

```
mov r2, #100
movw r1, #0x123
movt r1, #31
strb r1, [r2],#2
str r1, [r2,#-1]
strh r2,[r2],#3
```

Address	Value (in hex)
100	
101	
102	
103	
104	
105	
106	
107	

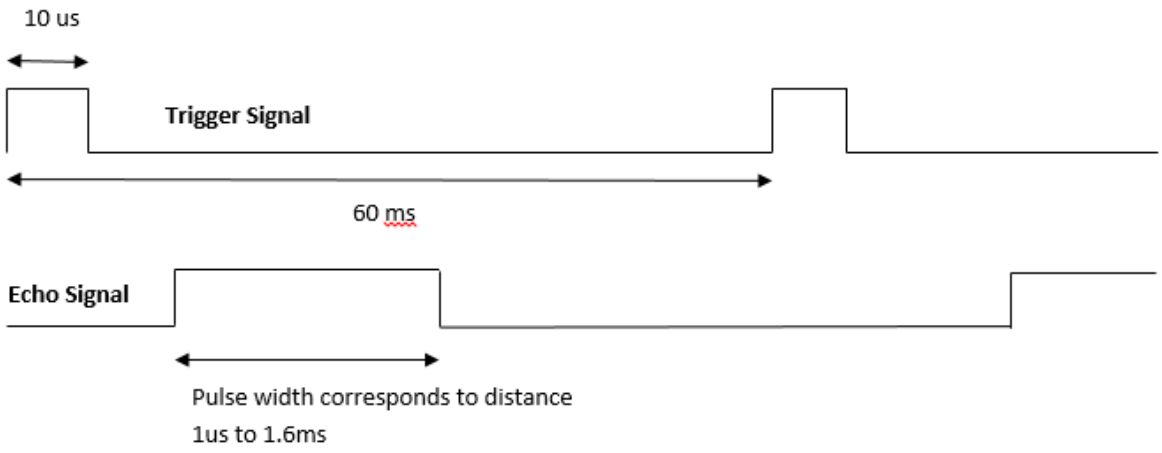
6. Write a C function “clear_and_enable” which takes a single external interrupt source as an argument. It clears the pending status for that register and enables interrupts from that source. It should return a 1 if the source was already enabled, otherwise it should return a 0. You need not check to see if x is a reasonable value. The last page references may be useful here. **[10 points]**

```
int clear_and_enable(int x)
{
```

Design Problem: Distance Measurement

Your task is to use a basic ultrasonic distance sensor (the HCS04) to measure distance using the SmartFusion kit. Be sure to read this entire problem before starting. APB read and write timing diagrams are provided on the last page of this exam for your reference.

Distance is determined by supplying a start pulse (trigger) and measuring the period of a subsequent echo pulse. The echo signal pulse will vary from 1us to 1.6ms if an echo is detected. Any value greater than 1.6ms means no target was detected. The distance can be determined by the scaling factor 2.54cm/us. An example timing diagram follows.



Throughout this problem, having shadow locations is fine.

Part 1: PWM Trigger Module **[17 points]**

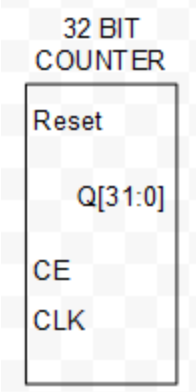
A Verilog-based PWM module will be used to provide the trigger signal. Although the trigger pulse width and period are fixed for this application, we want the trigger pulse width and period to be software configurable. Complete the following Verilog that implements the trigger signal with a 32-bit write-only pulse width register located at 0x4005000 and a 32-bit write-only period register located at 0x4005004. The pulse width and period should both be programmable from 1us to 100ms. Assume the system PCLK is 1 MHz.

Complete the Verilog which implements the trigger module.

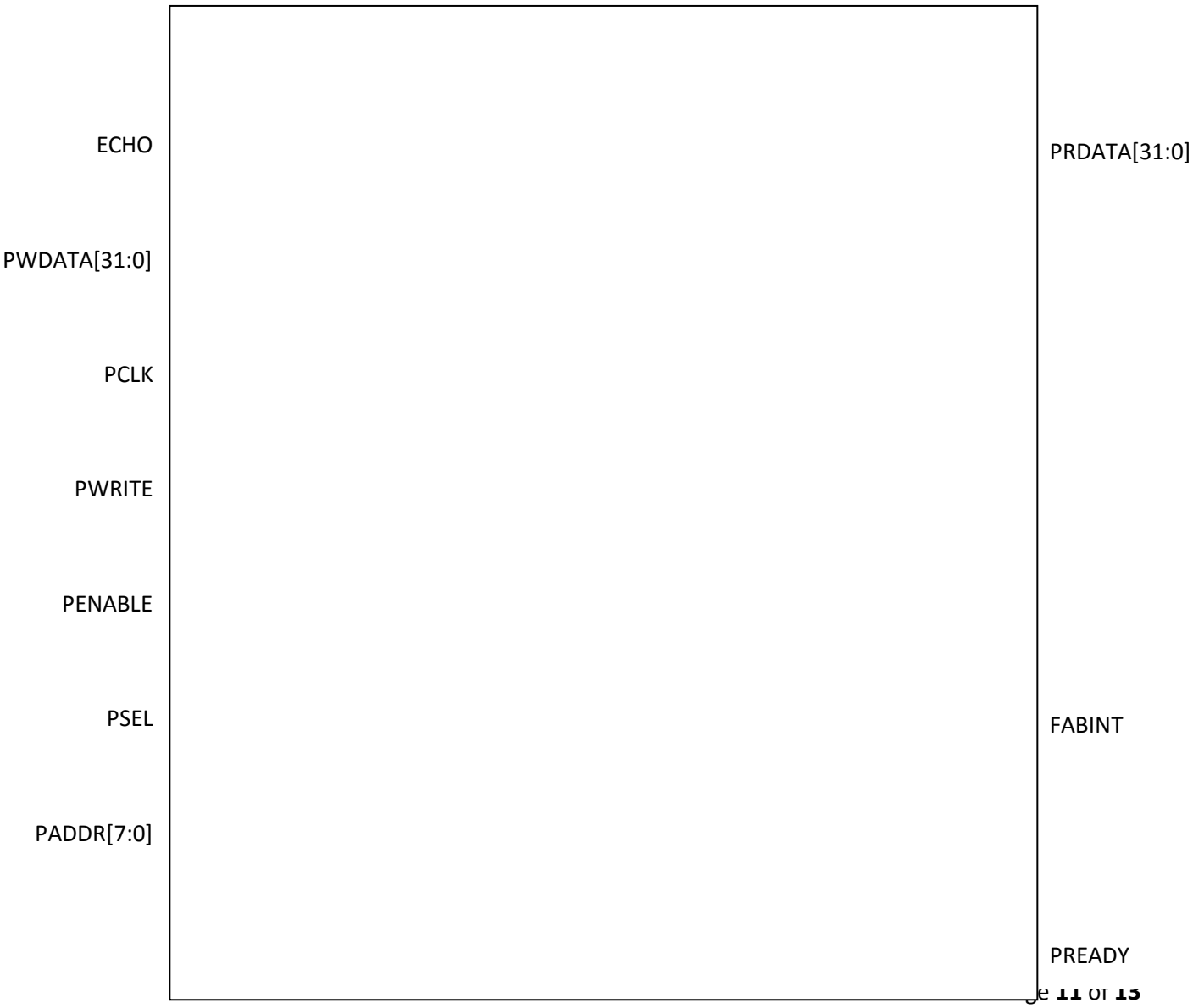
```
module trigger(
        input PCLK,
        input PRESERN,
        input PSEL,
        input PENABLE,
        input [7:0] PADDR,
        output PREADY,
        output PSLVERR,
        input PWRITE,
        input [31:0] PWDATA,
        output [31:0] PRDATA,
        output trigger );
```

Part 2: Echo Signal Measurement **[13 points]**

For this section you will design hardware module that will measure the pulse width of the echo signal. It needs to measure pulse width from 1us to 1.6ms. A reset signal will clear the measurement. The value should be read from APB3 bus as a word from location 0x4005000. A write to 0x40050008 will initiate the reset. Your logic should also provide a fabric interrupt signal when the measurement is done so an interrupt handler can be generated to read the value.



Given basic gates (AND, OR, INVERT), D Flip-Fops and 32-bit binary counters with reset and clock enable (figure on right), design the pulse measurement system. Assume the PCLK is 1 MHz. It is okay to use inverting bubbles and multiple inputs on gates. Assume the connections to the trigger module are made outside of this module.



Part 3: Interrupt Handler **[8 points]**

Write an interrupt handler generated by the end of measurement fabric interrupt that will read the pulse width measurement, convert it to meters (without rounding) and call a display function which will display the value. It should also reset the device you built in part 2. The prototype for the display function follows.

```
void display(float meters);  //meters is the number of meters away.
```

If the pulse measurement is outside the measurement range, you should display zero meters.

```
__attribute__ ((interrupt)) void Fabric_IRQHandler( void ) {
```

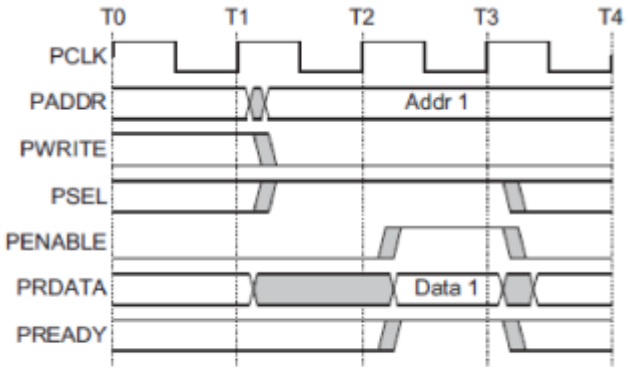


Figure 2-3 Read transfer with no wait states

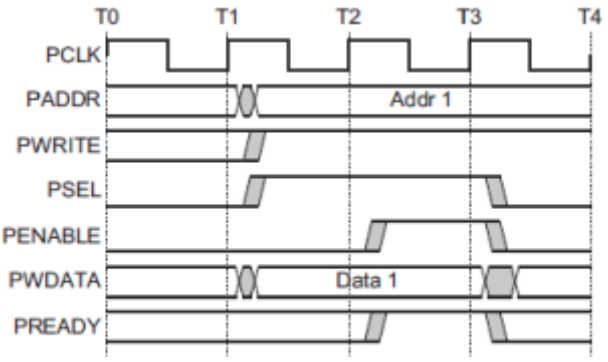


Figure 2-1 Write transfer with no wait states

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]

0xE000E100	SETENA0	R/W	0	Enable for external interrupt #0-31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) ... bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E180	CLRENA0	R/W	0	Clear enable for external interrupt #0-31 bit[0] for interrupt #0 bit[1] for interrupt #1 ... bit[31] for interrupt #31 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status
				... bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E280	CLRPEND0	R/W	0	Clear pending for external interrupt #0-31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) ... bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status