EEECS 373 Practice Midterm / Homework #3

Winter 2015

Due February 19th

Name: ________________________________ Uniquename: ____________

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

____

Problem #	Points
1	/10
2	/5
3	/20
4	/10
5	/15
6	/10
7	/15
8	/15
Total	**/100**

NOTES:

- **PUT YOU NAME/UNIQUENAME ON EVERY PAGE TO ENSURE CREDIT!**
- Can refer to the ARM Assembly Quick Ref. Guide and 1 page front/back cheat sheet
- Can use a basic/scientific calculator (but not a phone, PDA, or computer)
- Don't spend too much time on any one problem.
- You have 80 minutes for the exam.
- The exam is 9 pages long, including the cover sheet.
- Show your work and explain what you are doing. Partial credit w/o this is rare.

Name: ____________________________________ uniqname: ____________

1) **Fill-in-the-blank or circle the best answer. [10 pts, all or no points for each question]**

a) The ARM EABI specifies that registers __________ are caller-save.

b) The ARM Thumb-2 instruction set has ***16-bit / 32-bit / 16- and 32-bit*** encodings.

c) To generate 8 ns pulses, a PWM controller with a 50 MHz clock needs a ____ duty cycle.

d) An ARM EABI-compliant procedure that calls another procedure should ***always / sometimes / never*** save and restore the **`lr`** register.

e) ***UART / SPI / I2C*** supports flow control.

f) ***UART / SPI / I2C*** has built-in support for error checking by parity.

g) SPI bus transfers are ***asynchronous / synchronous*** and use ***dedicated chip select lines / addresses embedded in frames***.

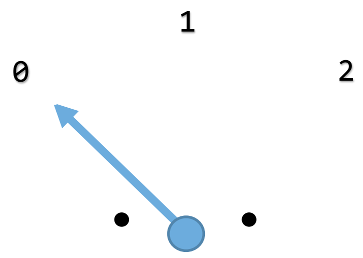
h) In the Verilog hardware description language, an `@(`_______________________`)` block would be used when implementing a flip-flop.

i) If multiple devices share a single interrupt line and generate interrupts at the same rate, then processor workload ***remains constant / grows linearly / grows quadratically*** with the number of devices.

j) By default, uninitialized global variables go in the ***.text / .data / .bss*** section.

Name: ______________________________________ uniqname: ____________

2) **Hardware Design. [5 pts]**



Imagine an arm with 3 positions: 0, 1, and 2. A motor controls this arm. The motor can be driven forward by asserting `MOTOR_FWD` and in reverse by asserting `MOTOR_REV`. It is an error to assert both signals.

The arm position is measured by an encoder that reports current arm position as an 8-bit value. Mechanical stops prevent the arm from going past 0 to the left or 255 to the right. The positions are at encoder values 20, 127, and 235. You may assume that the arm has no inertia, that is, if neither `MOTOR_FWD` nor `MOTOR_REV` are asserted, the `ARM_POSITION` will not change.

Write a hardware module to control this arm. Your module should move the arm to the `TARGET_POSITION` ***only*** when the `MOVE_REQ` signal is asserted. `POSITION` will not change while `MOVE_REQ` is high. Your module should assert `MOVE_DONE` when the arm is in the desired position. It should then wait until `MOVE_REQ` is de-asserted and de-assert `MOVE_DONE` in response. `MOVE_REQ` will not re-assert until `MOVE_DONE` is de-asserted.

```
module arm_control (
       input CLOCK,

       // control interface
       input [1:0] TARGET_POSITION,
       input MOVE_REQ,
       output MOVE_DONE,

       // arm interface
       input [7:0] ARM_POSITION
       output MOTOR_FWD,
       output MOTOR_REV
)

`define POS_0 8'd20
`define POS_1 8'd127
`define POS_2 8'd235
```

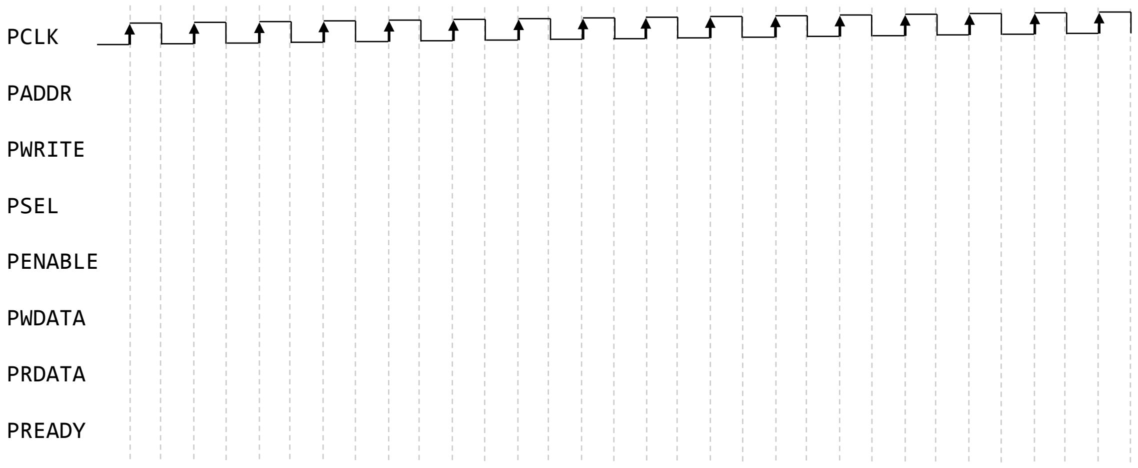
```
endmodule
```

Name: ___________________________________ uniqname: ____________

3) **Memory-Mapped I/O. [20 pts]**

Using basic circuit elements (e.g. logic gates and flops), sketch the glue logic required to interface the arm control module from Question 2 to the APB. Recall, the PSEL line is the peripheral select (i.e. it goes high when the processor is addressing the arm). The POSITION will be attached to data bits [1:0]. You should be able to change the arm position by writing to memory and read the current value by reading. A memory write should not return until the move is complete. Assume that PRDATA and PREADY are *shared* lines with other peripherals. The APB signals are: PCLK, PADDR, PWRITE, PSEL, PENABLE, PWDATA, PRDATA, and PREADY. You may ignore the motor I/O ports.

Sketch a timing diagram of an APB transaction that moves the arm from position 1 to position 2. Assume the arm peripheral is at address `0x20000080`. You only need to fill in relevant signals.



Name: ____________________________________ uniqname: ____________

4) **ARM Assembly Language. [10 pts]**

Encoding T1 ARMv7-M

`MOVT<c> <Rd>,#<imm16>`

15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0	15	14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
1	1	1	1	0	i	1	0	1	1	0	0	imm4	0	imm3	Rd	imm8

```
d = UInt(Rd);  imm16 = imm4:i:imm3:imm8;
if d IN {13,15} then UNPREDICTABLE;
```

Encoding T1 All versions of the Thumb instruction set.

`MOVS <Rd>,#<imm8>` Outside IT block.

`MOV<c> <Rd>,#<imm8>` Inside IT block.

15	14	13	12	11	10 9 8	7 6 5 4 3 2 1 0
0	0	1	0	0	Rd	imm8

```
d = UInt(Rd);  setflags = !InITBlock();  imm32 = ZeroExtend(imm8, 32);  carry = APSR.C;
```

Encoding T1 All versions of the Thumb instruction set.

`STR<c> <Rt>,[<Rn>,<Rm>]`

15	14	13	12	11	10	9	8 7 6	5 4 3	2 1 0
0	1	0	1	0	0	0	Rm	Rn	Rt

```
t = UInt(Rt);  n = UInt(Rn);  m = UInt(Rm);
index = TRUE;  add = TRUE;  wback = FALSE;
(shift_t, shift_n) = (SRType_LSL, 0);
```

a) What does the instruction `0x5088` do when executed?

b) Fill in machine code <u>in hex</u> around `0x5088` such that the arm peripheral from the previous question moves to position 1 when the instructions are executed.

`0x5088`

Name: ____________________________________ uniqname: ____________

5) **Assembly, C, and the ABI. [15]**

Recall the arm peripheral is at address 0x20000010 and encodes the arm position in the bottom two data bits.

a.) Write a C function, `bool check_move(int change)`, that takes a move and returns 1 if it is a possible move and 0 if the move cannot be made. The change argument describes the change of arm position, e.g. change "2" would move an arm from position 0 to position 0+2 = 2 and change "-1" would return an error if the arm were in position 0. This function should ***not*** move the arm.

b.) Write an assembly function `int move_arm(unsigned steps, bool reverse)` that moves the arm peripheral the requested number of steps forward or backward. Your function ***must*** call the `check_move()` function from (a) to verify if the requested move is valid before attempting to move the arm. `move_arm()` should return the current arm position, regardless of whether the arm moved or not.

The assembly code should assemble without errors. Write clearly and comment the code.

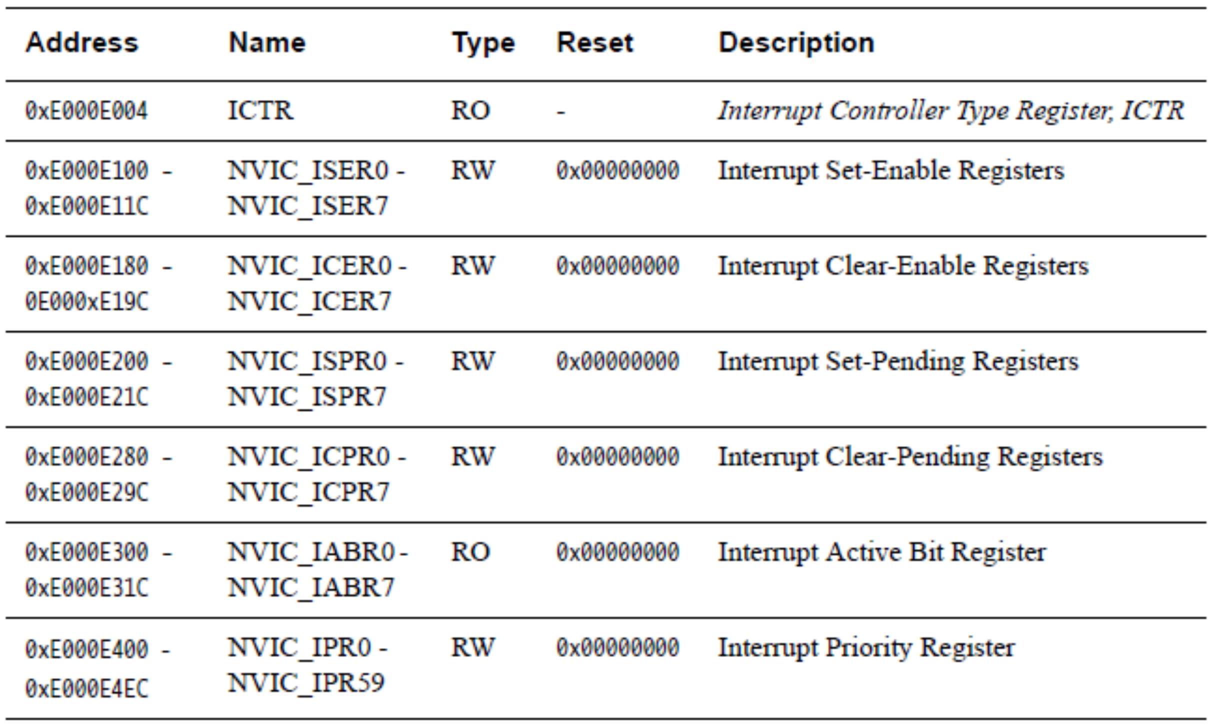
+1 EC for a 13-instruction solution. +3 EC for <13-instructions. This is tricky. Do not try until you are done with the exam. Use the last page and clearly indicate which we should grade if you try.

Name: ___________________________________ uniqname: ____________

6) **NVIC and Memory Map Comprehension. [10 pts]**

Write a C function `void` **`enable_interrupts(int x)`** that enables interrupt **`x`**. You need not check to validate that **`x`** is a legal interrupt number. The table below might be useful.

Table 6-1 NVIC registers



Address	Name	Type	Reset	Description
`0xE000E004`	ICTR	RO	-	*Interrupt Controller Type Register, ICTR*
`0xE000E100` - `0xE000E11C`	NVIC_ISER0 - NVIC_ISER7	RW	`0x00000000`	Interrupt Set-Enable Registers
`0xE000E180` - `0E000xE19C`	NVIC_ICER0 - NVIC_ICER7	RW	`0x00000000`	Interrupt Clear-Enable Registers
`0xE000E200` - `0xE000E21C`	NVIC_ISPR0 - NVIC_ISPR7	RW	`0x00000000`	Interrupt Set-Pending Registers
`0xE000E280` - `0xE000E29C`	NVIC_ICPR0 - NVIC_ICPR7	RW	`0x00000000`	Interrupt Clear-Pending Registers
`0xE000E300` - `0xE000E31C`	NVIC_IABR0 - NVIC_IABR7	RO	`0x00000000`	Interrupt Active Bit Register
`0xE000E400` - `0xE000E4EC`	NVIC_IPR0 - NVIC_IPR59	RW	`0x00000000`	Interrupt Priority Register

```
void enable_interrupts(int x) {








}
```

Name: ___________________________________ uniqname: ____________

7) **Startup and Interrupts [15 pts].**

In addition to the motor, the system has two buttons. Button0 is wired to Interrupt 7 and Button1 is wired to Interrupt 8. Whenever Button0 is pressed, the arm should move left (decrement position). If Button1 is pressed, the arm should move right (increment position).

Using the functions from Questions 5 and 6, write any remaining code required such that the arm goes to position 1 when the system powers on and the buttons behave as intended.

Name: ___________________________________ uniqname: ____________

8) **Logic Design [15 pts].**

The motor controller actually requires a PWM signal to operate. Given a 50% duty cycle, the motor will not move. At 70% it will drive forward, and at 30% it will drive in reverse. Design a digital circuit with inputs MOTOR_FWD and MOTOR_REV that takes the system clock of 100 MHz (CLKIN) and converts it to an output clock of 5 MHz with a 30, 50, or 70% duty cycle (MOTOR_CLK). You may use synchronously resettable D flip-flops and n-bit binary counters (make sure to specify the value of n). You may express combinational logic as Boolean expressions or using standard logic gates. You may assume that MOTOR_FWD and MOTOR_REV will never assert at the same time. Label things and write neatly.