EECS 373 *Midterm*

Winter 2017

Name: _____ unique name: _____

Sign the following honor code pledge.

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

_____

Scores:

Problem	Points
1	/12
2	/12
3	/12
4	/13
5	/12
6	/13
7	/13
8	/13
Total	**/100**

NOTES:

1. Closed book/notes. No calculators.
2. The references will be provided as a separate document.
3. Do not remove any pages from the test. We need to scan it.
4. No calculators, notes, books or resource not supplied as a part of the exam.
5. Don't spend too much time on any one problem.
6. The easy problems are worth as many points as the hard ones. Do the easy problems first.
7. You have 90 minutes for the exam.
8. Show work and make sure your reasoning is clear.

1) (12 pts.) Implement the following function using only NAND, NOR, and NOT gates. Your implementation should be minimal and two-level. Do not assume access to complemented input literals (e.g., you get a, b, and c as inputs, but not a', b', or c').

   f(a, b, c) = a'b'c + a'bc + abc + abc'

2) (12 pts.) For each description, fill in the blank with the letter associated with the correct tool.

   A. objdump
   B. as
   C. objcopy
   D. ld
   E. make
   F. gcc
   G. nm

   _____Combines object files and libraries into a single file, resolving symbols in the process.

   _____Converts human-readable assembly language code into machine-readable object files.

   _____Converts human-readable C language code into machine-readable object files.

   _____Displays instructions and data in object files in human-readable form.

   _____Displays symbol table in object or executable, including symbol resolution status.

   _____Translates object files from one format to another, potentially stripping debugging information and symbols in the process.

   _____Builds a presence and modification time based dependency graph transitively connecting build sources and build targets and executes build rules to traverse the graph, thus producing build targets.

3) (13 pts.) Blinky the clown needs a new blinking nose. You are tasked with designing a module to provide a signal for his new nose. Complete the following Verilog that implements the blinky module

with a 32-bit write only on-off MMIO register located at 0x40050000 and a 32-bit read and write period MMIO register located at 0x40050004.

The light has two states: off, or pulsing with a 50% duty cycle and a period between 500-1000ms. The period should be software configurable and the light should be turned off by default. The light is active low and the least significant bit of the on-off register controls the device (a register value of 0x1 should turn the light on). The system PCLK is 1MHz.

APB bus read and write timing diagrams are provided on the last page for your reference.

```
module blinky(
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 light)
```

Use this page to answer Problem 4.

4) (12 pts.) Fill in the blank.

a) Arguments that are passed into subroutines are placed in registers _____. If space is inadequate, the extra arguments are placed in_____.
b) The result of a subroutine is placed into register_____ or registers _____ and _____ before returning.
c) When a subroutine is called, the address that the subroutine should return to is placed in _____.
d) The PC holds the address of _____.
e) For this course, we are using ARM in _____ endian mode.
f) Registers _____ need to be saved by a subroutine before being used, while registers _____ need to be saved by a function before calling a subroutine.
g) To return from a subroutine, the assembly instruction _____ is used.

5) (13 pts.) Write an ARM assembly language procedure that implements the following C function in an EABI-compliant manner. Clearly comment your code. Label which register each value represents. Poorly commented code can result in a loss of points. Write your answer on the next page.

The function "conv" is an ABI compliant function with the following prototype.

uint32_t conv(uint32_t x);

```
uint32_t goblue(uint32_t x[], uint32_t y[], int n ) {
      int i;
      int sumx = 0;
      int sumy = 0;
      for(i=0; i<n; i++){
            if(x[i] > y[i])
                  sumx = conv(x[i])+sumx;
            else
                  sumy = conv(y[i])+sumy;
      }
      return (sumy - sumx);
}
```

Use this space to answer Problem 6.

6) (13 pts.) Your task is to output a 100 Hz PWM signal by writing 0x1 and 0x0 to GPIO 1, which is memory-mapped to at address 0x12345678. Your code must be written in C, and entirely implemented in an interrupt service routine (ISR) of a count-up timer with a 10 MHz clock (named Timer_ISR) that has both a timer compare register and timer overflow register, like that in Lab 5. The timer is already configured to fire interrupts on both compare and overflow events, and a status register (where the 0th bit signifies a compare event and the 1st bit signifies an overflow event) is located at 0x876543218. The timer compare register can be found at 0x87654320, and the timer overflow register can be found at 0x876543214.

At the end of each PWM period (every 1/100th of a second), please check register 0x45671234 for a duty cycle percentage (as an integer from 0 to 100). You do not have to worry about values outside of that range. If the duty cycle percentage changes, you should immediately change to outputting the new duty cycle percentage.

Register table map:
0x12345678: GPIO 1 (memory-mapped IO, PWM output)
0x45671234: PWM duty cycle percentage
0x876543210: Timer compare value
0x876543214: Timer overflow value
0x876543218: Timer interrupt status (If 0th bit is a 1, then the interrupt was a compare interrupt. if 1st bit is 1, then an overflow interrupt. They are guaranteed to not conflict.)

```
// triggers whenever the 10 MHz timer reaches the compare value
void Timer_ISR ( void ) {




}
```

7) (13 pts.) The following is a two-part problem on GPIOs and interrupts.

Part 1: In Lab2, you have seen how to initialize and set GPIO using assembly. Now you will implement the initGPIO in C rather than ARM assembly, and we provide you with setGPIO function for PART2. Both functions take in an int argument gpioNum as GPIO number.

initGPIO initializes GPIO pins to be either input or output, but not both, depending on the value of IO_mode argument. The function must return 0 if the target GPIO is already enabled for the requested mode, and return 1 if it is not.

setGPIO sets GPIO pins to be either high or low depending on the value of vLevel(set to high if vLevel is 1, set to low otherwise).

You don't need to check the sanity of the input (the input will always be valid), and assume we only have 32 GPIOs. If the GPIO pin is in output mode, setGPIO() will return 1 on success. If the GPIO pin is not in output mode, setGPIO() will return 0.

See the reference material at the end of the problem.

```
#define INPUT_MODE      0
#define OUTPUT_MODE     1
#define GPIO_IN                 0x40013084 // Read only bits for ports configured as inputs
#define GPIO_x_CFG              0x40013000 // GPIO configure register address

// NOTE: you don't need to implement this function.
// vLevel can only be 0 or 1. 0 means low voltage level and 1 means high voltage level
int setGPIO(int gpioNum, int vLevel);

// IO_mode can only be INPUT_MODE or OUTPUT_MODE
int initGPIO(int gpioNum, int IO_mode) {




}
```

Part 2: Now you should take advantage of the two functions in PART1 to write a fabric interrupt handler. Whenever the interrupt handler is invoked, you need to set GPIO 0 to input mode, wait for 1 ms, and read from GPIO 0 port, and output the opposite voltage level on GPIO 1. We have provided the function wait_1ms() to wait for exactly 1 ms, so you don't need to implement the waiting yourself. Assume other routines might change the mode of GPIO 1.

```
void wait_1ms( void ); // This function will wait for exactly 1 ms.

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



}
```

initGPIO task - In the case of the GPIO peripheral, we have two specific memory locations of interest. The first one is the configuration register. Note that these are not like the Cortex-M3 registers r0-r15. We just use the name register to indicate a specific location in memory. The microcontroller subsystem has 32 I/O lines, and each line has its own configuration register. The memory location starts at 0x40013000 for I/O line 0 (GPIO0). Each register is 32 bit long, and since the memory is byte indexed, we increment this memory location by 4 to find the configuration register for the other I/O lines, i.e., the configuration register for GPIO *i* is at *(0x40013000 + i*4)*.

The bits in the configuration register have a specific meaning. The following table is a summary of the different configuration possibilities. To see further description, look at the "Actel SmartFusion MSS User's Guide, Revision 1" on page 317.

Table 18-2 • **GPIO_x_CFG**

Bit Number	Name	R/W	Reset Value	Description
7:5	GPIOINT_TYPE	R/W	0b000	See Table 18-3 on page 318.
4	Reserved	R/W	0b0	Reserved
3	GPINTEN	R/W	0b0	0 = Interrupt disabled. 1= Interrupt enabled.
2	GPO_OUTBUFEN	R/W	0b0	0 = Disable output buffer if signal routed through IOMUX to IOBUF (dependent on IOMUX setting). 1 = Enable output buffer if signal routed through IOMUX to IOBUF (dependent on IOMUX setting).
1	GPINEN	R/W	0b0	0 = Input register disabled. 1 = Input register enabled.
0	GPOUTEN	R/W	0b0	0 = Output register disabled. 1 = Output register enabled.

As we want our I/O lines to be output, we will have to write 0x1 into the configuration registers.

setGPIO task - Once we configured all the I/O lines we need, we can set their status with the output register GPIO_OUT located at 0x40013088. Every bit inside this register represents one I/O line. Thus, clearing bit *i* to 0, will pull GPIO *i* low, while setting bit *i* to 1, will pull GPIO *i* high.

Table 21-2 • **SmartFusion Master Register Map (continued)**

Register Name	Address	R/W	Reset Value	Description
"GPIO Register Map"				
GPIO_x_CFG (x = 0)	0x40013000	R/W	0x0	GPIO Configuration register for bit 0
⋮	⋮	R/W	⋮	⋮
GPIO_x_CFG (x = 31)	0x4001307C	R/W	0x0	GPIO Configuration register for bit 31
GPIO_IRQ	0x40013080	R/W	0x0	Interrupt Status Register
GPIO_IN	0x40013084	R	0x0	Read only bits for ports configured as inputs
GPIO_OUT	0x40013088	R/W	0x0	Read/write bits for ports configured as outputs

If you use this page for answers, write “See last page.” next to the relevant problem(s) and mark the problem number(s) clearly on this page.