# Final Exam

**Date:** December 15, 2017

Printed Name: ________________________________ ________________________________
Last, First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam. *You will not reveal the contents of this exam to others who are taking the makeup thereby giving them an undue advantage*:

Signature: ____________________________________________________________

**Instructions:**

- ***Write your UT EID on all pages (at the top) and circle your instructor's name at the bottom.***
- Closed book and closed notes. No books, no papers, no data sheets (other than the last four pages of this Exam)
- No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. *Anything outside the boxes will be ignored in grading*.
- You have 180 minutes, so allocate your time accordingly.
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
- Unless otherwise stated, make all I/O accesses friendly.
- *Please read the entire exam before starting.* ***See supplement pages for Device I/O registers.***

| | | |
|---|---|---|
| **Problem 1** | 10 | |
| **Problem 2** | 10 | |
| **Problem 3** | 10 | |
| **Problem 4** | 15 | |
| **Problem 5** | 15 | |
| **Problem 6** | 15 | |
| **Problem 7** | 15 | |
| **Problem 8** | 10 | |
| **Total** | 100 | |

**(10) Question 1.** Place one letter in the box for each question that represents the best answer.
A) To force the compiler to not optimize the access
B) To place the variable in nonvolatile ROM
C) To force the variable to be placed in a register
D) To place the variable in volatile RAM
E) To make the variable private to the file
F) To make the variable private to the function
G) Because of Arm Architecture Procedure Call Standard
H) To force the variable to be placed on the stack

**(1) Part a)** Why do we add **`static`** to an otherwise local variable? ............................................... [ ]

**(1) Part b)** Why do we add **`const`** to an otherwise global variable? ............................................... [ ]

**(1) Part c)** Why do we add **`static`** to an otherwise global variable? ............................................... [ ]

**(1) Part d)** Why do we use a local variable? ............................................................................ [ ]

I) To execute instructions faster
J) Because of the Central Limit Theorem
K) To reduce the latency of other interrupts
L) To save power, making the battery last longer
M) To prevent Aliasing
N) Because of the Nyquist Theorem
O) To decouple the execution of the ISR with the main program
P) To reduce noise and improve signal to noise ratio
Q) To synchronize one computer to another
R) To improve bandwidth

**(1) Part e)** Why do we place a FIFO queue between an ISR that reads data from an input port................... and the main program that processes the data? [ ]

**(1) Part f)** Why does a Harvard architecture have two (or more) buses? ............................................ [ ]

**(1) Part g)** Why should the time to execute an ISR be as short as possible? ....................................... [ ]

**(1) Part h)** Why would you ever wish to use the PLL and slow down the bus clock .............................. so the software runs slower? [ ]

**(1) Part i)** Why would we use hardware averaging on the ADC? ................................................... [ ]

**(1) Part j)** Why does the UART protocol use start and stop bits? .................................................. [ ]

**(10) Question 2:** For parts a) to f), please place one **numerical value** in the box for each question.

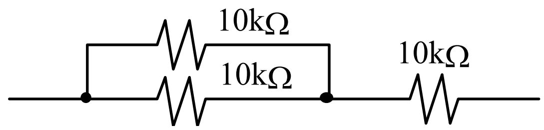
**(2) Part a)** You are measuring heart rate using a bioelectric signal and need frequency components from 1 to 10 Hz. The ADC precision is 8 bits. UART baud rate is 100,000 bps. What is the …………. slowest sampling rate (sampling frequency) possible?

**(1) Part b)** A fixed-point number system has a resolution of 0.1 cm. The integer part is 2, …………….. what is the value of the fixed-point number? …

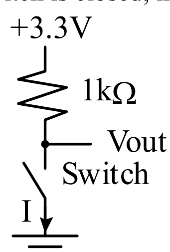
**(1) Part c)** The bit time is 1ms. What is the baud rate? ……………………………………

**(1) Part d)** A fixed-point number system has a resolution of $2^{-2}$ cm (0.25 cm). What integer …………. do we use to represent the value -3.75 cm?

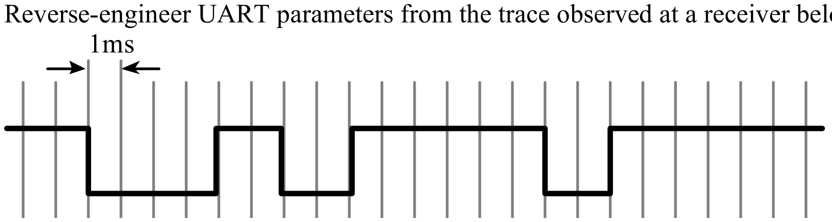
**(1) Part e)** These three resistors could be replaced by one resistor. Give the resistance value of this one equivalent resistor? ………………………………………………………………………………




**(1) Part f)** How much current (I) flows when the switch is closed, include units ………………………




Reverse-engineer UART parameters from the trace observed at a receiver below.




**(2) Part g)** What is the *data value* transferred over the UART in **unsigned decimal**? …………

**(2) Part h)** What is the *baud rate* in **bits/sec**? ………………………………………………………

**(10) Question 3:** *Interrupt*

**(6) Part a)** Complete the assembly subroutine that initializes SysTick to interrupt every 1 sec. The bus clock is 16 MHz; that each bus cycle is 62.5 ns. The interrupt occurs when the counter goes from 1 to 0. Initialize CTRL to 7. The goal is to toggle PB1 every 3600 seconds. You may assume PB1 is already initialized to be a GPIO output. Fill in the blanks as needed

```
       .data
       .align 2
[                                                        ]

       .text
Init:   LDR   R1,=SysTick_LOAD
[                                                        ]
       STR   R0,[R1]          // establish interrupt period

       LDR   R1,=SysTick_CTRL
[                                                        ]
       STR   R2,[R1]          // arm SysTick interrupts
[
                                       //clear bit 0 of PRIMASK
]
       BX    LR
```

**(4) Part b)** Write the SysTick ISR in assembly that toggles PB1 every 3600 seconds. Write 2 to GPIOB_DOUTTGL31_0 to toggle PB1. You cannot read GPIOB_DOUTTGL31_0.

```
SysTick_Handler:
```

**(15) Question 4:** *FSM*

**(5) Part a)** Draw a Moore finite state graph with 2 inputs (SW1, SW0) and one output (LOCK). Initially, the LOCK is high (1). Any time both inputs are simultaneously high, the machine reverts to the initial state with the LOCK high. Call this initial state **Init**. The sequence just SW0 high (input=1), both switches low (input=0), and then just SW1 high (input=2) will cause the LOCK to go low (0). All other states have the LOCK high. Assume the FSM runs at a fixed rate of 100 Hz periodically performing output, wait 10ms, input, and next operations. Use **pointer addressing** to access the next state.

**(10) Part b)** Show the C code, including the struct that defines your finite state graph in ROM. Define a state pointer **Pt**.

The execution engine in the main program is given and cannot be changed.

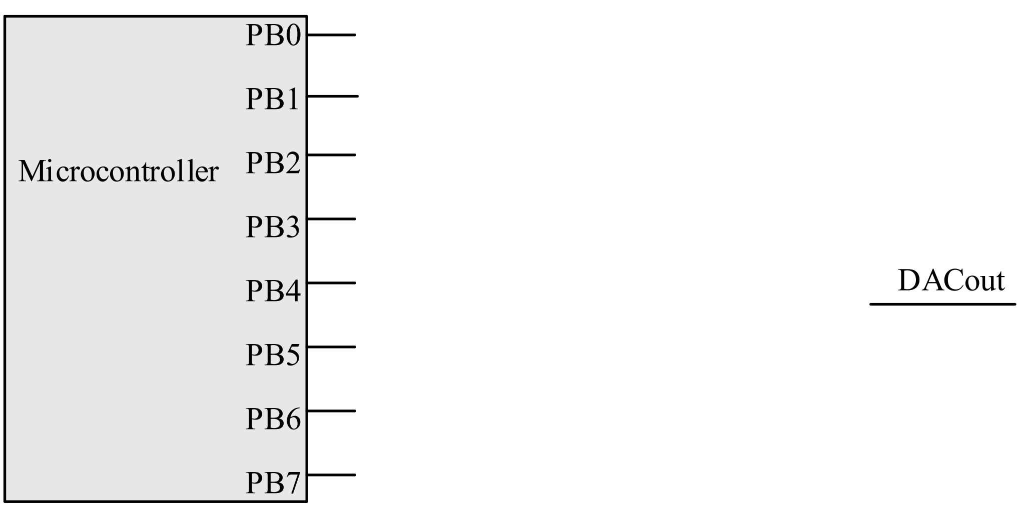
```
int main(void){uint32_t input;
  Port_Init();    // Port A and Port B initializations are given (do not write)
  SysTick_Init(); // SysTick initialization is given
  Pt = Init;      // initial state (not to write)
  while(1){
    GPIOB->DOUT_31_0 = Pt->Out;    // set LOCK to 0 or 1
    SysTick_Wait10ms(1);           // fixed delay
    input = GPIOA->DIN31_0&0x03;   // read switches: 0,1,2,3
    Pt = Pt->Next[input];  }}}
```

**(15) Question 5:** *Interfacing*

**(5) Part a)** Interface an LED to the microcontroller PA7 output. If PA7 is high the LED should be on. Assume the desired operating point is 2.1V 2mA. Assume the output low voltage of the MSPM0 is 0.2V. Assume the output high voltage of the MSPM0 is 3.1V. Show the circuit, including resistor values.

**(5) Part b)** Design a circuit with two switches and one LED. Assume the switches are ideal, and the LED operating point is 1.3V 2 mA. There is no microcontroller in this solution. The LED should be on if both switches are pressed; otherwise, the LED should be off. Show the circuit, including the switches, the LED, and resistors as needed (include resistance values).

**(5) Part c)** Design an 8-bit DAC using multiple resistors; interface it to Port B. Show the circuit, including resistor values. Give actual resistor values that will work.



**(15) Question 6:** *FIFO queue.* You are asked to implement a 16-bit FIFO that can handle up to 19 elements. You cannot add additional global variables. You cannot change the function prototypes. You **must use pointers** to access the FIFO

```
uint16_t FIFO[20];
uint16_t *Gpt,*Ppt;
```

**(3) Part a)** Write the routine that initializes the FIFO, making the FIFO empty.

```
void Fifo_Init(void){ // Initialize FIFO

}
```

**(6) Part b)** Write the routine that puts data into the FIFO. If the FIFO is full, this routine should wait until there is room in FIFO for the data. You can add local variables, but no statics or globals.

```
void Fifo_Put(uint16_t data){ // enter data into FIFO

}
```

**(6) Part c)** Write the routine that gets data from the FIFO. If the FIFO is empty, this routine should wait until there is data in FIFO to return. You can add local variables, but no statics or globals.

```
uint16_t Fifo_Get(void){ // if empty spin until there is data

}
```

**(15) Question 7:** *Local variables*
**(3) Part a)** What does the function **Func** do?

**(7) Part b)** Implement the operation performed by this assembly function in C. The assembly is written in AAPCS. Use good names for parameters and local variable(s).

```
//R0, pt points to an array
//R1, the size of array
Func: PUSH {R0,R1}
      MOVS R2,#0     ;result
      PUSH {R2,R7}
      MOV  R7,SP
//***this point for part c)***
loop: LDR  R0,[R7,#8]
      LDRH R1,[R0]
      ADDS R0,#2
      STR  R0,[R7,#8]
      LDRH R0,[R7,#0]
      EORS R0,R0,R1
      STRH R0,[R7,#0]
      LDRB R0,[R7,#12]
      SUBS R0,#1
      STRB R0,[R7,#12]
      BNE  loop
      LDRH R0,[R7,#0]
      ADD  SP,#4
      POP  {R7}
      ADD  SP,#8
      BX   LR
```

**(5) Part c)** Assume at the beginning of the execution of the function **Func**, the stack is empty, SP equals 0x20200400, R0 equals 0x20201000 (pointer to an array), R1 equals 4 (size of the array), R2 equals 2 (some random irrelevant value), and R7 equals 11 (some random irrelevant value). Show the contents of the stack at the point in the execution of **Func** signified by the *****.** Also specify the value of R7 by drawing an arrow into the stack figure.

| | |
|---|---|
| 0x202003E4 | |
| 0x202003E8 | |
| 0x202003EC | |
| 0x202003F0 | |
| 0x202003F4 | |
| 0x202003F8 | |
| 0x202003FC | |
| 0x20200400 | |
| 0x20200404 | |

**(10) Question 8**: *UART interrupt*

Assume you have the SSI interface to the LCD you used in Lab 5. In C, this LCD output data function is

```
void SPI_OutData(char data){
  while((SPI1->STAT&0x02) == 0x00){}; // spin if TxFifo full
  GPIOA->DOUTSET31_0 = 1<<13;         // RS=PA13=1 for data
  SPI1->TXDATA = data;
}
```

The overall goal of the communication is to transfer all data from the UART0 receiver to the LCD. Assume the UART0 is initialized for receiver interrupts. When there is data in the receiver FIFO, RXRIS is set and a UART interrupt is triggered.

**(6) Part a)** Show the UART ISR that reads one 8-bit data from the receiver, acknowledges the interrupt and sends the data to the LCD using the SSI port. To read data, you read from **UART0->RXDATA**.

```
void UART0_IRQHandler(void){ uint32_t status;
  status = UART0->CPU_INT.IIDX; // reading clears bit in RIS
  if(status == 0x01){   // 0x01 receive timeout
```

**(2) Part b)** The hardware automatically pushes R0, R1, R2, R3, R12, LR, PC, and PSW on the stack when the interrupt is triggered. These registers are automatically popped at the end of the ISR by the BX LR instruction. Is the ISR code allowed to use the other registers R4-R11? Choose the best answer, placing A-F in the box.

A) No, since these registers are not saved, using these registers would cause errors in the main program.
B) Yes, according to AAPCS, R4-R11 are reserved for interrupts, so they can be freely used.
C) Yes, according to AAPCS, any function can use R4-R11 if it first saves the registers and then restores them afterward.
D) No, according to AAPCS, R4-R11 are reserved for the main program, so they cannot be used in an ISR.
E) No, according to AAPCS, the stack must be aligned to 8 bytes.
F) Yes, according to AAPCS, R4-R11 are automatically saved by the compiler for every function.

**(1) Part c)** Does the hardware automatically disable interrupts during the execution of the ISR? Choose the best answer, placing A-F in the box.

A) Yes, the execution of an interrupt is more important than the execution of the main program.
B) Yes, disabling interrupts prevents the execution of one ISR from being interrupted by itself
C) Yes, according to AAPCS, the hardware automatically disables interrupts during the execution of the ISRs.
D) No, but according to AAPCS, the software automatically disables interrupts during the execution of the ISRs.
E) No, interrupts are not disabled to allow interrupts with a lower priority (higher value in priority register) to interrupt.
F) No, interrupts are not disabled to allow interrupts with a higher priority (lower value in priority register) to interrupt.