Final Exam

Date: Dec 10, 2015

UT EID: ________________________

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:

- 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	12	
Problem 3	12	
Problem 4	12	
Problem 5	12	
Problem 6	12	
Problem 7	10	
Problem 8	20	
Total	100	

(10) Question 1. Please place **one letter/number** for each box. Choose the best answer to each question.

Part i) What is the consequence of running the microcontroller at a higher speed? ....

Part ii) When is a 7406 or ULN2003B driver needed in interfacing and LED? ....

Part iii) What is the advantage of using binary vs. decimal fixed point? ....

Part iv) What does the prototype **void fun(uint32_t *)** say? ....

Part v) What memory does code get placed in? ....

Part vi) Why is the bandwidth always lower than the baud-rate? ....

Part vii) How do we make a declaration be placed in ROM? ....

Part viii) The AAPCS convention requires that we maintain the contents of these registers inside a subroutine? ....

Part ix) Where are local variables allocated their space? ....

Part x) Which two shift operations are the same? ....

A) The Cortex M has a Harvard Architecture.
B) The PC always fetches instructions from flash memory in a von Neumann architecture.
C) ASL and LSL are the same.
D) ASR and LSR are the same.
E) In RAM because code needs to be able to grow the stack.
F) Registers R0 through R3.
G) Registers R4 through R11.
H) In ROM because it does not get modified.
I) UART uses start and stop bits in addition to the 8-bits of data.
J) Local variables are stored on the stack.
K) Local variables are stored in registers.
L) The LED needs more than 3.3 V.
M) The LED needs more than 6 mA on the MSPM0.
N) Binary takes less space than decimal.
O) It creates a negative logic interface.
P) To satisfy the Nyquist Theorem.
Q) Binary fixed-point math is simpler than decimal.
R) Because the UART sends a data bit value 0 as 0V and a data bit value 1 at 3.3V.
S) The function does not accept an 8 or 16 bit unsigned integer.
T) The receiver uses it to synchronize timing with the transmitter.
U) The function expects a pointer to a 32-bit unsigned integer as input.
V) Black box testing is more detailed than white box testing.
W) It decouples the production of data from the consumption of data.
X) The switch needs more than 10mA current.
Y) If running on battery we drain the battery faster.
Z) It provides debugging, allowing you to download code and debug your software.
1) In order to handle either positive or negative values.
2) By using a static modifier in the declaration.
3) By using a const modifier in the declaration.
4) By using a volatile modifier in the declaration.
5) To tell the compiler the subroutine should not change its value.
6) Specifies it as an address or a pointer.

(12) Question 2

(7) Part a) R0 has x, R1 has y, R2 has z. All variables are unsigned 32-bits. What conditional branch instruction is needed for each?

Instructions	C code

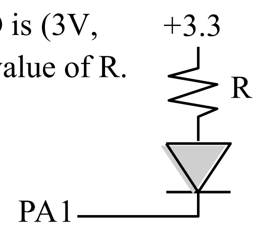
`MOVS R3,R0`
`ORRS R3,R1`
`     Skip1`
`MOVS R2,#1`
`Skip1:`
	`if(x|y){`
	`  z = 1;`
	`}`

`CMP  R0,R1`
`     Skip2`
`MOVS R2,#2`
`Skip2:`
	`if(x < y){`
	`  z = 2;`
	`}`

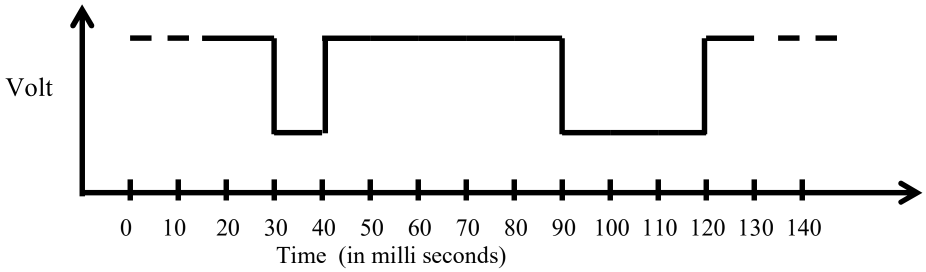
`Loop3:`
`ADDS R0,#1`
`CMP  R0,R1`
`     Loop3`
	`do{`
	`  x++;`
	`}while(x < y);`

(2) Part b) A repeating pulse signal that is ON for 5ms and OFF for 5ms is being sampled with an ADC. What should the sampling frequency be to faithfully capture the signal?

(3) Part c) In the LED circuit interface to the right the operating point of the LED is (3V, 5mA) and the V_{OL}=0.5V. Will the interface work? If it works, then calculate the value of R. If it does not work, explain why?



(12) Question 3. Reverse-engineer UART parameters from the trace observed at a receiver below.



(3) Part a) What is the **integer** *data value* transferred over the UART?

(3) Part b) What is the *baud rate* in **bits/sec**?

(2) Part c) What is the *maximum bandwidth* in **bytes per second**?

(4) Part d) Assume the UART0 has been initialized with busy wait synchronization. Write a C function that writes one character to the UART0.

(12) Question 4.

a) (4 points) For an 8-bit ADC with an analog input voltage range of 0 to 2.55V, what are the following:

(i) ADC precision

(ii) ADC range

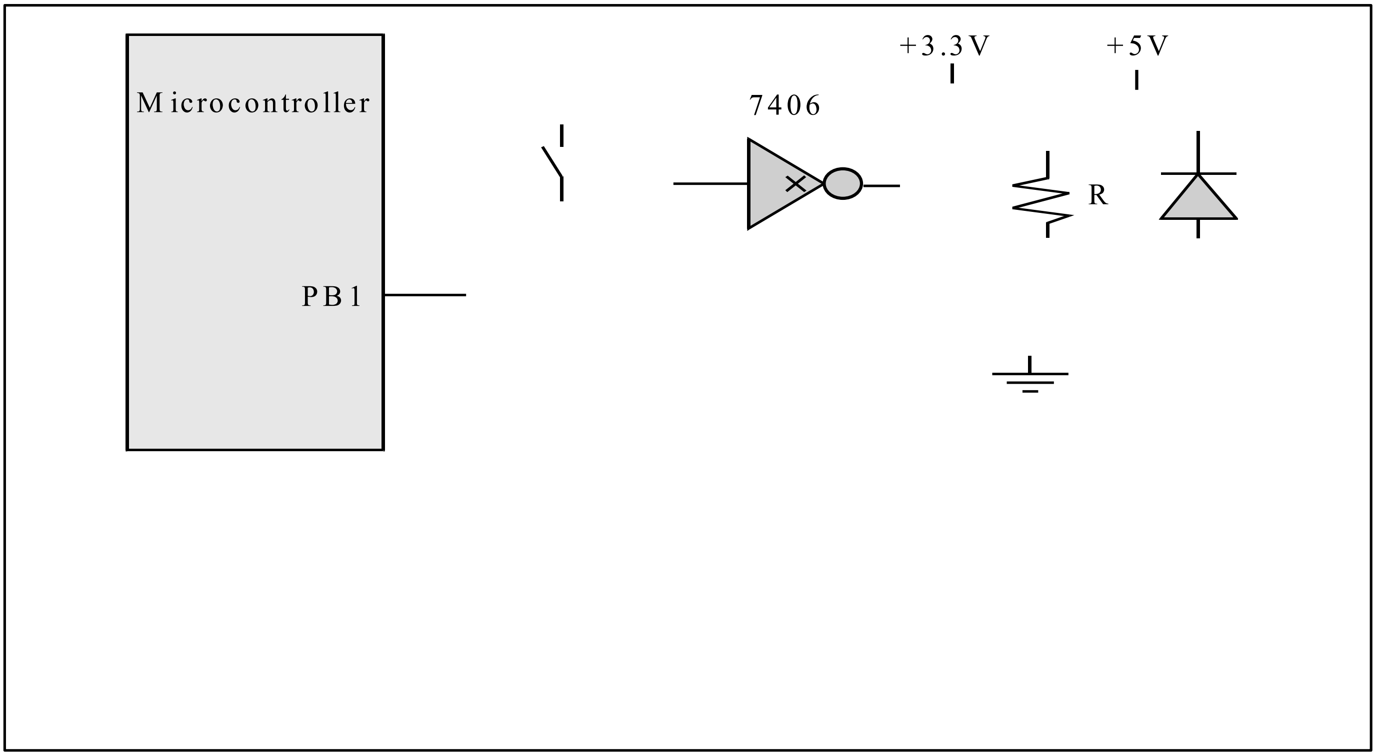
(iii) ADC resolution

b) (2 points) What will the above 8-bit ADC return if the input voltage is 1.0V?

c) (6 points) An external ADC is interfaced to Port B. PB8 is an output; write 1, then write 0 to start the ADC. PB9 is an input; it will read 0 when started, and read 1 when conversion is complete. PB7-0 are inputs containing the ADC digital result. the ADC. Write an ***ADC_In*** function (in C) that uses busy-wait synchronization to sample the ADC. The function starts the ADC, waits for the conversion, reads the ADC data, and returns the 8-bit binary number. Assume the GPIOB pins are initialized.

```
uint8_t ADC_In(void) {
```

(12) Question 5.
The desired operating point of an LED is 1V, 3mA. Interface this LED to PB1 using positive logic. Assume the microcontroller output voltages are V_{OH} = 3.2V and V_{OL} = 0.2V. Specify values for any resistors needed. Show equations of your calculations used to select resistor values.
Draw the circuit along with any additional components needed.



(12) Question 7: FIFO
Write C programs that implement a FIFO data structure with exactly two elements. You have to implement the **`FiFo_Init, FiFo_Get`** and **`Fifo_Put`** functions. The FiFo declaration and function prototypes for the functions are given below. Feel free to add any global variables you need:

```
// Declarations
char FiFo[2];
// Add any other declarations here
```

```
//Prototypes
void FiFo_Init();         // Initializes the FiFo variables
unit8_t FiFo_Put(char data);  // Adds data to Fifo returns 0/1
                              // for failure/success
unit8_t FiFo_Get(char *data); // Copies to *data from Fifo
                              //returns 0/1 for failure/success
```

(10) Question 8: Convert the C code into assembly, assuming the AAPCS parameter passing convention and local variables. Remember, local variables use the stack, not registers. (You MUST use local variables on the stack).

```
uint16_t pow(unit8_t base, uint8_t exp){
  uint16_t prod;
  uint16_t n;

  prod = 1;
  for(n=exp; n>0; n--) {
    prod = prod * base;
  }
  return prod;
}
```

(20) Question 9: **(Program)** A stepper motor can be controlled by writing a 4-bit number to it. The repeating sequence 5,6,10,9,5,6,10,9… moves it clockwise (CW) and the repeating sequence 5,9,10,6,5,9,10,6… moves it counter clockwise (CCW). The delay between writes determines the speed of the motor. Assume a constant speed of the motor with time between writes of 50ms. Design a Moore FSM with four states that takes a single input PA0 (0: CW; 1:CCW) and four outputs (PB0-3).

a) (10 points) Give the FSM state graph for the stepper motor.

b) (10 points) Complete the code below by adding state #defines and FSM array entries and the FSM loop.

```
struct State{
  uint8_t out;    // output produced in this state
  uint32_t wait; // delay in 10µs units;
                 // Can call delayus(count) to wait for count*10us
  uint8_t next[4]; // list of next states
};
typedef struct State state_t;
state_t FSM[4] = {





}
uint8_t curState = ; //set the initial state here
int main() {
      // All Port Initialization done for you - Complete the FSM loop below
      …
      while(1){




     }
}
```