Final Exam

Date: May 14, 2015

UT EID: _____ Circle one: MT, NT, JV, RY, VJR

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	10	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	10	
Problem 7	12	
Problem 8	8	
Problem 9	20	
Total	100	

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

Part i) Why do we sometimes use the phase lock loop? ……. [ ]

Part ii) Why did we use the open collector 7406 gate to interface the LED? ……. [ ]

Part iii) Why did we use fixed-point to represent measured distance? ……. [ ]

Part iv) Why did we dump input/output data into buffers in Lab 4? ……. [ ]

Part v) Why do we put programs in flash memory? ……. [ ]

Part vi) Why does the UART use start and stop bits? ……. [ ]

Part Vii) Why do we specify a global variable as **`static`**? ……. [ ]

Part viii) Why do we specify a local variable as **`static`**? ……. [ ]

Part ix) Why do the I/O definitions have **`volatile`** in the definitions? ……. [ ]

Part x) Why do we specify a function parameter as **`const`**? ……. [ ]

A) The Cortex M has a Harvard Architecture.
B) The PC always fetches instructions from flash memory in a von Neumann architecture.
C) Some machine instructions are 16 bits and others are 32 bits.
D) It reduces the scope of the data making the data private to the file.
E) The Cortex M processor on the TM4C123 does not support floating point operations.
F) The left/right shift is faster than multiply/divide.
G) In order to represent non-integer values.
H) To create bounded latency and provide for real-time operation.
I) It is nonintrusive debugging.
J) It is minimally intrusive debugging.
K) The interface must control both voltage and current so the LED is the proper brightness.
L) The LED needs more than 3.3 V.
M) The LED needs more than 8 mA.
N) Buffers can store more data than can be printed using the UART.
O) It creates a negative logic interface.
P) To satisfy the Nyquist Theorem.
Q) It illustrates to our client how the program works.
R) Because the UART sends a data bit value 0 as 0V and a data bit value 1 at 3.3V.
S) Message can vary in length and it is used signify the end of the message.
T) The receiver uses it to synchronize timing with the transmitter.
U) It provides a mechanism to minimize bandwidth.
V) Black box testing is more detailed than white box testing.
W) It decouples the production of data from the consumption of data.
X) It provides for ceiling and floor.
Y) If we slow down processor execution, it will save power. If we execute faster, we do more processing.
Z) It provides for debugging, allowing you to download code and debug your software.
1) In order to handle either positive or negative values.
2) To allocate it in RAM, making it persistent across subroutine calls.
3) To allocate it in ROM, and ROM is nonvolatile.
4) To tell the compiler to fetch a new value each time it is accessed.
5) To tell the compiler the subroutine should not change its value.
6) Specifies it as an address or a pointer.

(10) Question 2

(2) Part a) What are the addressing modes used in the following ARM instructions?

Instructions	Addressing Modes
`MOV R0, #10`	
`LDR R0, [R1]`	
`BL sublabel`	
`ADD R2, R1`	
`PUSH {R4-R11, LR}`	

(2) Part b) In order to specify the desired baud rate for a bus clock frequency is 80 MHz, the divider has been correctly calculated as 50.125. What values should the **UART0_IBRD_R** and the **UART0_FBRD_R** registers be initialized to?

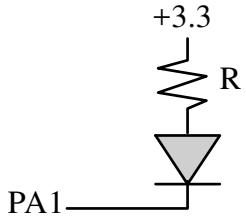
`UART0_IBRD_R =` ☐

`UART_FBRD_R =` ☐

(2) Part c) If the ADC sampling frequency is 100 Hz, what range of frequencies in the analog input can safely be represented in the digital samples?

☐

(2) Part d) Consider an LED with a desired operating point of (I_d,V_d). Let V_{OL} V_{OH} I_{OL} and I_{OH} be the operating parameters of the digital output on PA1. What is the design equation needed to calculate the desired resistance R for this circuit?

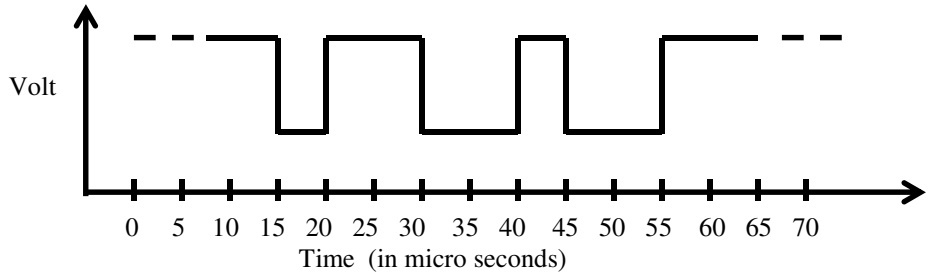


☐

(2) Part e) What is the relationship between the range, precision and resolution of an ADC, given that the sampling frequency is f?

☐

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



(2) Part a) What is the *data value* transferred over the UART in **hexadecimal**?

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

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

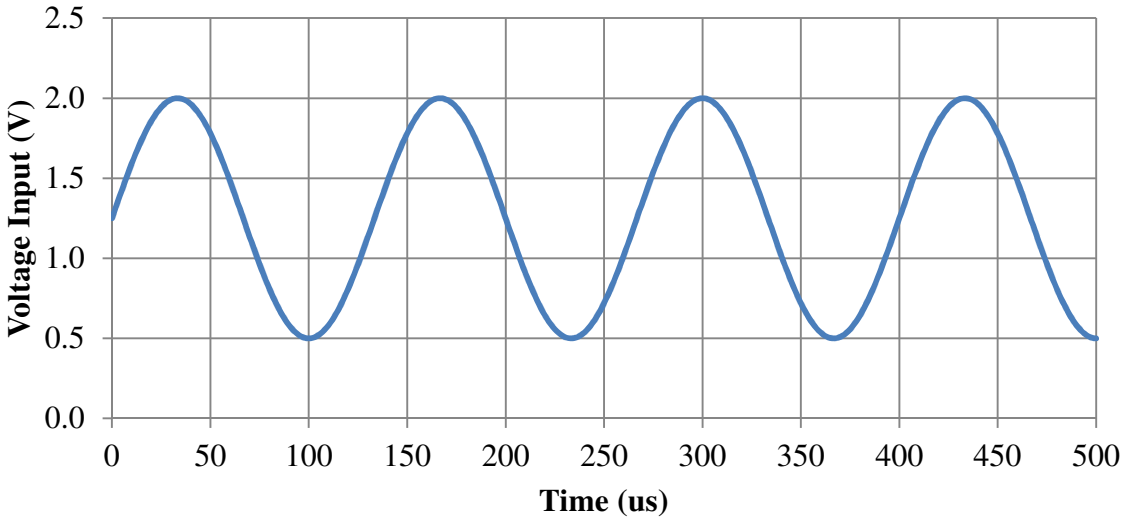
(3) Part d) Assume the UART has been initialized with busy wait synchronization. Write a C function that reads one character from the UART.

(2) Part e) Assume the receiver software uses busy-wait synchronization. The main program reads all the data available from the UART and then processes the data. The maximum time required to process the data is 125us. Is it possible to lose data? If so, explain how to change the UART so no data is lost. If no data can be lost, explain how the UART works so no data are lost.

(10) Question 4. Analog Devices AD7641 is an 18-bit, 0 to 2.5V range, 2MSPS SAR ADC. A student is attempting to capture a sinusoid signal of frequency 7.5 kHz using the AD7641. Using the 18-bit ADC and periodic interrupt, he programs the system to interrupt at a frequency of 20 kHz. Each time the system interrupts, he calls **`AD7641_In()`** to get one sample of the signal from the AD7641.

(2) Part a) If the AD7641 input is 1.25 V, what will be the digital value **in hex** returned by this ADC?

(4) Part b) Assuming the first sample is taken at time t=0, mark the (time, voltage) points on the plot below specifying the data collected by the ADC. Just mark the points, you do not need to calculate the values.



(4) Part c) Is it possible to recreate the original signal from the captured samples? If your answer is *yes*, explain how. If your answer is *no*, what is the term used to refer to this loss of information?

(10) Question 5. You will design an embedded system using a Moore FSM. There are two inputs (PB3, PB2) and two outputs (PB1, PB0). The FSM runs in the background with 1 kHz SysTick periodic interrupts. Initially both outputs will be low, and you may also assume both inputs are initially low. *If PB3 rises before PB2 rises, then set PB1 high. If PB2 rises before PB3 rises, then set PB0 high. If both rise during the same 1-ms window, set both PB1 and PB0 high. After either or both PB1 and/or PB0 become high, let the output remain fixed.* The initial state is s=0.

(4) Part a) Show the FSM graph in Moore format. Full credit for the solution with the fewest states.

(6) Part b) The **`struct`** and the main program are fixed. Show the C code that places the FSM in ROM, and write the SysTick ISR. **`PORTB_Init`** initializes PB3-PB0 and makes the outputs low. **`SysTick_Init`** initializes interrupts at 1 kHz. **`PORTB_Init`** and **`SysTick_Init`** are given. Full credit awarded for friendly access and good programming style. The initial state will be s=0.

```
const struct State{
  uint32_t out;
  uint32_t next[4];
};
typedef const struct State State_t;
uint32_t s;  // state number
```

```
void main(void){ PORTB_Init();
  s = 0; // initial state
  SysTick_Init();
  EnableInterrupts();
  while(1){}}
void SysTick_Handler(void){
```

```
// Initialize array of states
```

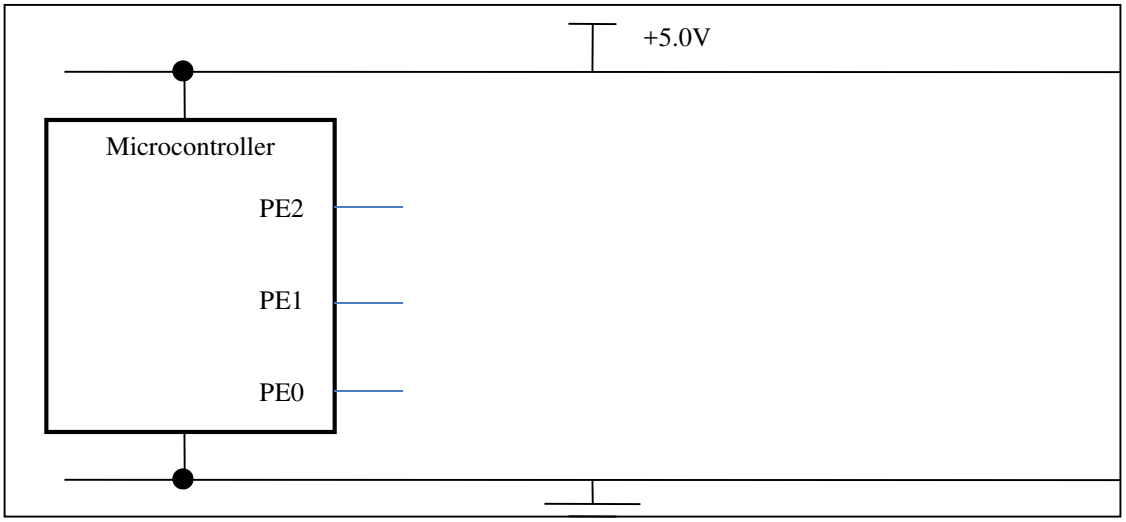
```
// read input


// change state


// friendly write output


}
```

(10) Question 6. (a): You are given two 1 kΩ resistors and four 2 kΩ resistors. Build a 3-bit DAC circuit (connected to PE2, PE1, PE0) using *all* resistors. Complete the table below where a few of the input logic voltage values at PE2, PE1, PE0 are shown. Calculate the output voltage *Vout* of the DAC for those input values given that V_{OH} = 5.0V, and V_{OL} = 0V.



PE2	PE1	PE0	Vout
0	0	0	
0	0	1	
0	1	0	
1	0	0	

(b) The output of the DAC circuit you built in part (a) is now connected to a speaker whose resistance is very low and can be approximated to be 0 Ω. The other end of the speaker is grounded. Calculate the current through the speaker when the logic voltage values at PE2, PE1, PE0 are 100. Show your work

(12) Question 7: FIFO

(2) Part a) What is the difference between *FIFO* and *Mailbox*?

(10) Part b) Write a C program that implements FIFO using two stack data structures. You have to implement the **`Fifo_Get`** function using two stacks. **`Fifo_Put`** is already given to you. Return a value of -1 if the FIFO is empty. The stack data functions are given to you, having *push*, *pop* and *empty* functions that you must use. The function prototypes for these functions are given below.

```
// Prototypes of the stack functions that you can use
// Assume stacks do not overflow (infinite size)
int  pop1();        // Gets the element at the top of the stack1
void push1(int);    // Puts the element at the top of the stack1
int  empty1();      // Returns 1 if stack1 is empty, 0 otherwise
int  pop2();        // Gets the element at the top of the stack2
void push2(int);    // Puts the element at the top of the stack2
int  empty2();      // Returns 1 if stack2 is empty, 0 otherwise

// Put an element into the back of the FIFO.
// data is never –1 (the error code)
void Fifo_Put(int data) {
  push1(data); // pushes element data onto stack1
}
```

```
// Get the element at the head of the FIFO
int Fifo_Get(void) {









}
```

(8) Question 8: Convert the C code into assembly, using local variable allocation phases. Remember, local variables use the stack, not registers. Put exactly one assembly line into each box.

```
; *****binding phase***************
```

```
uint16_t calc(void){
  uint16_t sum;

  uint16_t n;

  sum = 0;

  for(n=255; n>0; n--) {
    sum=sum+n;
  }

  return sum;
}
```

```
; 1)*****allocation phase *********
calc PUSH {R4,LR}
                          ;allocate 4 bytes

; 2)******access phase ************
      MOV  R0,#0
                                  ;sum=0
      MOV  R1,#255
                                  ;n=255
loop                              ;R1=n
                                  ;R0=sum
      ADD   R0,R1                 ;R0=sum+n
                                  ;sum=sum+n
                                  ;R1=n
      SUBS R1,#1                  ;n-1
                                  ;n=n-1
      BNE  loop

; 3)******deallocation phase *****

      POP  {R4,PC}  ;R0=sum
```

(20) Question 9: **(Program)** Many of you could not play your ideal music for Lab 10. Valvano had enough with people asking for Full licenses on Keil. He knows sound files are the source of the problem, they are simply too big. You are told to change the coding of the sound files so the resulting array packs two 4-bit samples into one byte, resulting in a compression ratio of 2:1. All wav files are converted to 4-bit samples at 8 kHz. For example the first ten 4-bit samples of the **start** sound (see below) are 8,9,9,10,11,11,12,14,15,15. Notice 8,9 are "packed" into the byte 0x89. During play time, the packed values are decompressed into their original form and sent to the DAC. The game has 4 sounds, each in a different array named **startS, shootS, deathS** and **quietS**, each of a different length. The following code declares the constants, variables and structure used in the solution. Read the code carefully and answer the below questions.

```
#define start 0
#define shoot 1
#define death 2
#define quiet 3

const uint8_t startS[450] = {0x89,0x9A,0xBB,0xCE,0xFF … };
const uint8_t shootS[280] = { … };
const uint8_t deathS[8]   = {0x8B,0xDE,0xFE,0xDB,0x85,0x32,0x12,0x35};
const uint8_t quietS[1]   = {0x88};
struct sound{
  uint32_t length;         // number of bytes in the array
  const uint8_t *samples;  // pointer to the array
};
typedef struct sound Sound_t;

// sounds is the array of structs one per sound
Sound_t sounds[4] = {{450,startS},{280,shootS},{8,deathS},{1,quietS}};

uint32_t cSound; // holds the current sound number (0,1,2,or 3)

// Declare any other globals you need here
```

Your task is to write the following three routines, along with any globals you need above:

```
// Setup SysTick so it interrupts periodically at 8 kHz, bus=80MHz
void SysTick_Init(void){




  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;// Priority 2


}
```

```
// Sound is always playing. Call this function to change the sound
// Called with a single input which is 0-3 specifying which sound to play
// Start playing the new sound from the beginning after switching.
void ChangeSound(uint8_t soundNum){









}
```

```
// SysTick_Handler calls DAC_Out output one 4-bit value
// cSound specifies the sound to play
// loop current sound when the end is reached
// DAC_Out is given, you do not need to write it
void SysTickHandler(void){

















}
```