Final Exam

Date: Dec 11, 2014

UT EID: _____ Circle one: ME, JV, RY

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	20	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	10	
Problem 7	20	
Problem 8	10	
Total	100	

(20) Question 1 (Miscellaneous)

(3) Part a) If an *Interrupt Service Routine* modifies register R0, the ISR does not have to save and restore R0, why?

R0,R1,R2,R3,R12,LR,PC,PSW are automatically pushed on the stack when the interrupt occurs and restored by the BX LR at the end of the ISR

(5) Part b) A DAC is used to output a *sine wave* using SysTick Interrupts and a sine-wave table. Assume the DAC has 7 bits, the DAC output is connected to a speaker, the SysTick ISR executes at 32kHz, the sine table has 256 elements, and one DAC output occurs each interrupt. The DAC output range is 0 to 3.3V. The bus clock is 80 MHz. The ADC maximum rate is 125 kHz. What frequency sound is produced, in Hz?

One output occurs each SysTick ISR, so the sine wave output will be 32kHz/256 = 125 Hz.

(3) Part c) A DAC has a *range* of 0 to 3V and needs a *resolution* of 1mV. How many bits are required? In other words, what is the *smallest number of DAC bits* that would satisfy the requirements?

precision is (3-0)/0.001 = 3000 alternatives. 11 bits would only be 2048 alternatives, so we need to use 12 bits to get 4096 alternatives.

(3) Part d) An embedded system will use an ADC to capture electrocardiogram (EKG) data. The frequency range of the human EKG spans from 0.1 Hz to 100 Hz. What is the *slowest rate* at which we could sample the ADC and still have a faithful representation of the EKG in the digital samples? Give your answer as the time between samples.

Nyquist Theorem says sampling rate needs to be larger than $2*f_{max}$, so $f_s > 200$ Hz, which means the time between samples must be less than 1/200 = 5ms.

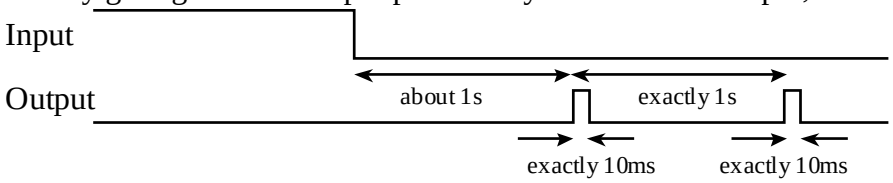
(3) Part e) An 8-bit ADC (different from the TM4C123) has an input range of 0 to +10 volts and an output range of 0 to 255. What *digital value* will be returned when an input of +7.5 volts is sampled? Give your answer as a decimal number.

256*7.5/10 = 192 or 255*7.5/10 = 191

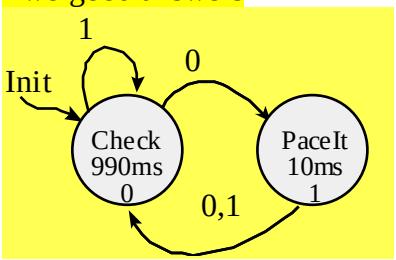
(3) Part f) A serial port (UART1) is configured with one start, 8 data bits, one stop and a baud rate of 50,000 bits/sec. What is the *maximum possible bandwidth* of this port in bytes/sec?

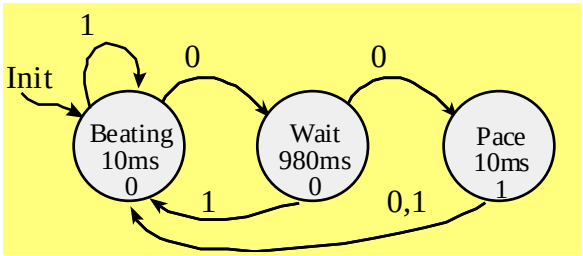
50,000 bits/sec (8 bits data/10 bits in frame)*(1byte/8bits) = 5,000 bytes/sec

(10) Question 2 (FSM). You will design a pacemaker using a Moore FSM. There is one input and one output. The input will be high if the heart is beating on its own. The input will be low if the heart is not beating on its own. If the heart is not beating your machine should pace the heart. If the heart is beating on its own, the input will be high and your output should be low. However, if the input is low, you should pace the heart by giving a 10 ms output pulse every 1 sec. PB0 is output, PB1 is input.



(5) Part a) Show the FSM graph in Moore format. Full credit for the solution with the fewest states.
Two good answers





Pace must be 10ms, but the other two can be any times that add to 990ms

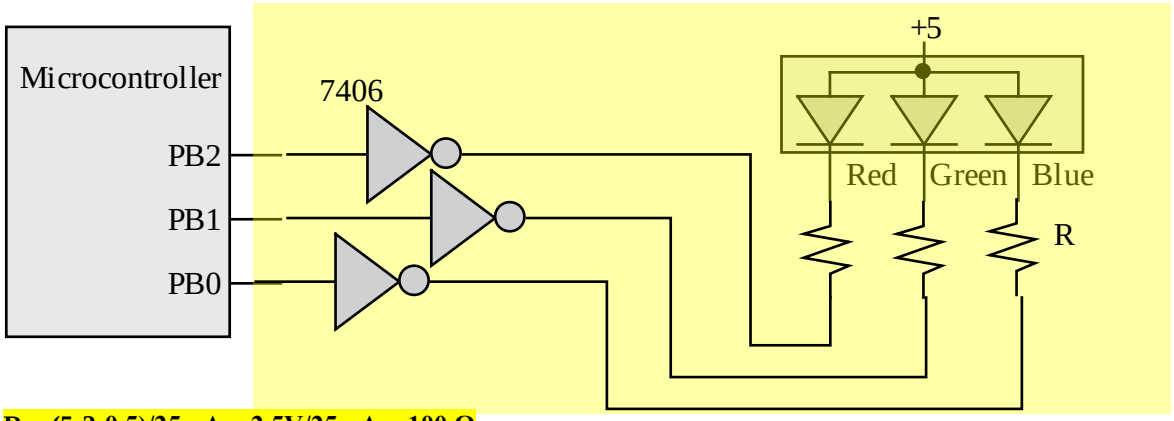
(5) Part b) The structure and the main program is fixed. Show the C code that places the FSM in ROM, and specify the initial state in the box.

```
const struct State{
      uint32_t out;
      uint32_t wait;
      uint32_t next[2];
};
typedef const struct State State_t;
uint32_t s;
#define Check  0
#define PaceIt 1
STyp FSM[2]={
 {0, 990, { Wait,   Check}},
 {1,  10, { PaceIt, Check}}};
```

```
void main(void){ PORTB_Init();
  SysTick_Init();
  s = Check ;
  while(1){
    GPIO_PORTB_DATA_R = FSM[s].out;
    SysTick_Wait1ms(FSM[s].wait);
    Input = (GPIO_PORTB_DATA_R&0x02)>>1;
    s = FSM[s].next[Input];
  }}
```

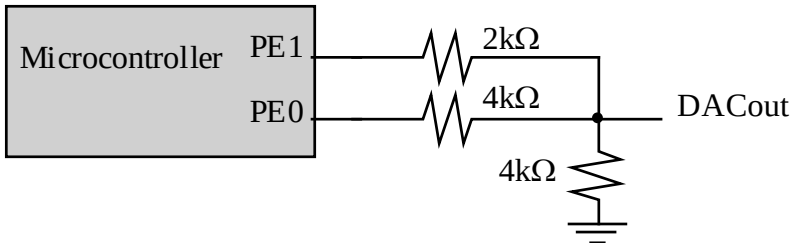
```
#define Beating 0
#define Wait    1
#define Pace    2
STyp FSM[3]={
 {0,  10, {   Wait, Beating}},
 {0, 980, {   Pace, Beating}},
 {1,  10, {Beating, Beating}}};
```

(10) Question 3. Interface a multicolor LED to the microcontroller. Each color is controlled by a separate diode with an operating point of 2V, 25mA. You can use any number of 7406 inverters, and any number of resistors. Assume the V_{OL} of the 7406 is 0.5V. Assume the microcontroller output voltages are V_{OH} = 3.0V and V_{OL} = 0.1V. Specify values for any resistors needed. Show equations of your calculations used to select resistor values. Make each output control one color, positive logic.



R = (5-2-0.5)/25mA = 2.5V/25mA = 100 Ω

(10) Question 4 (DAC). What are the maximum voltage, precision, and resolution of this DAC? Assume the microcontroller output voltages are V_{OH} = 3.2V and V_{OL} = 0.0V.



Maximum voltage	**Precision**	**Resolution**
3*3.2/4, 0 to 2.4V	2 bits or 4 alternatives	3.2V/4 =0.8V

Solve for output 01_2, 2k in parallel with 4k is (2*4/(2+4))= 1.333k. 3.2V-4000-1.333-ground,
so DACout is 3.2V*1.333/5.333= 0.8
Solve for output 10_2, 4k in parallel with 4k is 2k. 3.2V-2000-2000-ground,
so DACout is 3.2V*2000/4000= 1.6
Max output is 11_2, 0.8+1.6=2.4V

(10) Question 5 (UART).

(5) Part a) Write a C function that sends an 8-byte message using UART1. Assume the UART1 is already initialized for busy-wait synchronization. The 7 bytes of payload are passed by reference to your function. You will send an 8th byte that is an error-checking code (ECC), which will be the **exclusive or** of the previous 7 bytes. If you call a function, you must define that function.

```
// Input: str is a pointer to a 7-byte array of data to be transmitted
void UART1_OutMessage(const uint8_t str[7]){
  uint8_t ECC=0;
  int i;
  for(i=0; i<7; i++){
    while((UART1_FR_R&0x0020) != 0);  // wait until TXFF is 0
    UART0_DR_R = str[i];
    ECC = ECC^str[i];
  }
  while((UART1_FR_R&0x0020) != 0);  // wait until TXFF is 0
  UART0_DR_R = ECC;
}
// Input: 8-bit data to be transmitted
void UART1_OutChar(const uint8_t data){
  while((UART1_FR_R&0x0020) != 0);  // wait until TXFF is 0
  UART0_DR_R = data;
}
```

(5) Part b) Assume you have received the 8-byte message from the UART1 on the other microcontroller, and the message has been placed in global RAM as an array of 8 bytes.

`Message SPACE 8`

Write an assembly subroutine that checks the ECC to determine if an error has occurred. Return R0=0 if the message is ok, and return R0≠0 if the ECC does not match. The subroutine will access the global array called `Message`. *If you **exclusive or** all 8 bytes, you should you get 0.*

```
CheckMessage
     MOV  R0, #0    ; start with zero
     MOV  R1, #8    ; counter
     LDR  R2, =Message  ; pointer to array
Loop LDRB R3, [R2]
     EOR  R0,R0,R3  ; XOR all bytes
     ADD  R2, #1    ; next address
     SUBS R1, #1    ; loop counter
     BNE  LOOP
     BX   LR
```

(10) Question 6 (debugging). Consider the code, which declares one global data structure and implements two functions that manipulate the structure. Note that the code has many bugs.
(8) Part a) Write down as many bugs as you can find and propose a solution. Use the boxes below to describe each bug, the lines affected by it (possibly multiple lines with same type of bug), and a solution. If a bug is missing lines of code, mark down the two line numbers between which your solution code needs to be inserted and just write the extra code in the "Solution" column. For example, lines 6 and 11 are missing a semi-colon as marked below. *You are not allowed to delete lines.*

```
1:  #include <stdint.h>
2:  struct fifo {
3:    char    data[512];
4:    uint8_t x, y, z;};
5:  typedef struct fifo fifo_t;
6:  fifo_t myData
7:  int8_t Fifo_Put(char c) {
8:    if (myData.z == 512) {
9:      return(-1);
10:   }
11:   myData.data[myData.x] = c
12:   myData.x = [myData.x + 1] % 511;
13:   myData.z = myData.z + 1;
14: }
15: int8_t Fifo_Get(char* c) {
16:   c = myData ->data[myData->y];
17:   myData->y = (myData->y + 1) % 511;
18:   myData->z = myData->z + 1;
19: }
```

Line(s)	Description	Solution
6, 11	Missing ;	Add ;
4	Precision too low	uint16_t
12	Wrong brackets	(my_data.x + 1)
12, 17	Wrong operator	& 511
12, 17	Alternative, wrong value	% 512
13-14	Exit with no return value	return(0);
16	Assigning address rather than value	*c = …
16, 17, 18	Wrong operator	-> should be .
15 – 16	Missing empty test	if (my_data.z == 0) return(-1);
18 – 19	Missing return	return(0);
18	Size decrease	+1 to -1

```
struct Elem {
char    data[512];
uint16_t putI, getI, size;};
typedef struct Elem fifo_t;
fifo_t myData;
int8_t Put(char c) {
  if (myData.size == 512) {
    return(-1);  }
  myData.data[myData.putI] = c;
  myData.putI = (myData.putI+1)&511;
  myData.size = myData.size + 1;
  return 0;}
```

```
int8_t Get(char* c) {
  if(myData.size == 0) return -1;
  *c = my_fifo.data[myData.getI];
  myData.getI = (myData.getI+1)&511;
  myData.size = myData.size - 1;
  return 0;
 }
```

(2) Part b) What is the purpose of the return value of the function **`f()`** and the function **`g()`**?

Return error codes. If put fails because it is full, return -1. If get fails because it is empty, return -1.

(20) Question 7 (Local Variables, AAPCS and Parameter-passing)
Answer the questions that follow with reference to the code given below. Assume initially that R3=3, R4=4, R5=5, R11=11, and LR = 0x30F.

```
// Main.c
extern uint32_t Func(  uint32_t param, uint16_t  *pt  ); //[1]

int main(void){
  uint16_t glob;
  uint32_t param = 14;

  glob=42;
  param = Func(param,&glob);   //[A]
  glob += param;               //[C]
}
```

```
main PUSH  {r3-r5,lr}
     MOVS  r4,#0x0E
     MOVS  r0,#0x2A
     STR   r0,[sp,#0x00]
     MOV   r1,sp
     MOV   r0,r4
     BL    Func
     MOV   r4,r0
     LDRH  r0,[sp,#0x00]
     ADD   r0,r0,r4
     UXTH  r0,r0
     STR   r0,[sp,#0x00]
     MOVS  r0,#0x00
     POP   {r3-r5,pc}
```

```
; Func.s
     EXPORT  Func          ;[2]
     AREA    |.text|, CODE, READONLY, ALIGN=2
     THUMB

loc equ 8                  ; Binding [3]

Func
    SUB SP,#4
    PUSH {R2,R11}
    LDRH R2,[R1]
    ADD R2,#1
    STRH R2,[R1]
    STR R2,[SP,#loc]
    ADD R0,R2
    STR R0,[SP,#loc]
    POP {R2,R11}                ;[B]
    ADD SP,#4
    BX  LR

    ALIGN
    END
```

a) (5 points) Compete the three missing blanks in lines labeled [1], [2] and [3].
b) (2 points) Circle the **Allocation** and **Deallocation** steps for the local variable **`loc`**.
c) (3 points) Assuming the SP is initialized to `0x20000400`. What are the contents of the Stack (and the value of the SP) after **`main`** calls **`Func`** and the instruction at [B] has been completed.

	0x200003E4	
	0x200003E8	
SP ->	0x200003EC	57
	0x200003F0	43
	0x200003F4	4
	0x200003F8	5
	0x200003FC	0x0000030F
	0x20000400	
	0x20000004	

d) (5 points) What is the value of the variable **glob**:

I. After instruction at [A] is completed?

43

II. After instruction at [C] is completed?

100

e) (5 points) Give the C equivalent of the assembly code corresponding to the subroutine **Func**.

```
uint32_t Func(uint32_t param, uint32_t *pt){
      uint32_t loc;

      (*pt)++;
      loc = *pt;

      loc += param;
      return(loc);
}
```

(10) Question 8: **(assembly/C)** The left and right sides represent corresponding C and ARM assembly (think of the assembly as the compiler-produced code of the C part). Both side contain a few missing lines, which you are to fill in. Each box below represents exactly one missing line of code (in either C or ASM). Note that the missing C lines are placed to roughly correspond to the existing lines in the ASM program and vice versa.