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).***
- 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 120 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.***

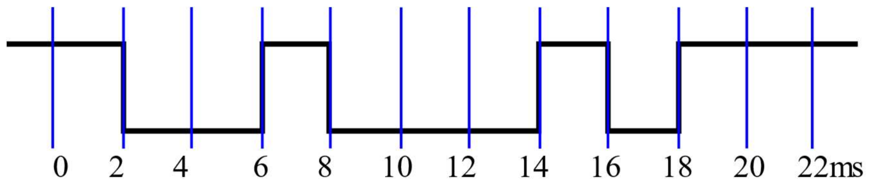
(5) Problem 1) Allocation. Consider this piece of C code. For each item, specify where it is located.

A) Permanent in RAM B) Permanent in nonvolatile ROM C) Temporary in R7
D) Temporary in R0 E) On the stack F) Not stored at all on computer

```
#define SIZE 10 -----------------------------------------------SIZE-

static uint32_t *pt; ----------------------------------------pt-----

uint32_t Operate(uint32_t y){ -------------------------y-----

  uint32_t buffer[SIZE]; --------------------------------buffer--

  return 0; ----------------------------------------------------------0-----

}
```

(5) Question 2) UART. The following UART transmission was recorded. It is one frame.

(3) Part a) What is the baud rate? Give units.



(2) Part b) What is the 8-bit data value in hex? --------------------------------

(15) Question 3) UART output. It is possible to communicate between microcontrollers using the UART protocol even if the microcontroller does not have UART hardware. You will write C language function to implement the transmitter using PB7. You may assume PB7 is already initialized as a regular GPIO output pin. To implement the transmitter, you simply write the **start, b0, b1, b2, b3, b4, b5, b6, b7, stop** bits to the PB7 pin with a fixed delay. This is called bit-banging. Assume the baud rate **BR** is the same as your answer to Q2 part a). To create the baud rate, you are given a **Delay** subroutine that delays exactly 1/**BR**. You will call the **Delay** subroutine 10 times, once for each bit of the frame. Make sure the idle condition (after the stop and before the next start) is correct. How to check your answer: let **z** be the 8-bit data value for your answer to Q2 part b). If you were to call this **UART_OutChar(z)** the output on PB7 should be the same as the figure in Q2. You should NOT be accessing any UART registers, just writing to **GPIOB->DOUT31_0.**

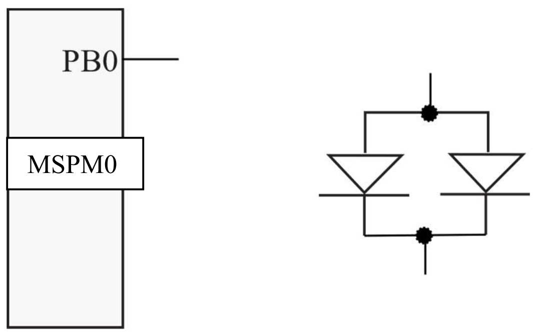
```
// GPIOB->DOUT31_0 outputs to all 32 bits of GPIOB
void UART_OutChar(uint8_t data){
```

(10) Problem 4) Equations. Give the functional equations in terms of these parameters:

V_{OL}, output low voltage of MSPM0 in volts
V, voltage across the LED in volts
R, resistance of a smallest resistor of the DAC in ohms
I, current through the LED in amps
n, number of bits in the DAC
b, baud rate of the UART in bits/sec
max, the maximum possible DAC voltage in volts
min, the minimum possible DAC voltage in volts
f_{min}, minimum sound frequency you wish to create with the DAC and interrupts
f_{max}, maximum sound frequency you wish to create with the DAC and interrupts
f, the MSPM0 bus frequency in Hz

(3) Part a) Give the relationship for the **power** dissipated in an LED.	Power =	Units of power =
(3) Part b) Largest possible interrupt period; the ISR outputs one value to the DAC, creating sound.	Period =	Units of period =
(4) Part c) Give the relationship for the DAC **resolution**.	Resolution =	Units of resolution =

(10) Question 5) LED interface. Interface these two LEDs to PB0 with one resistor. The two LEDs are connected so that if the software outputs a 0 to PB0, then both LEDs should turn on. If the software outputs a 1 to PB0, both LEDs should turn off. Assume V_{OH} of PB0 is 3.1V, and V_{OL} of PB0 is 0.3V. The operating point of each LED is 1V, 2mA. Show the circuit and include the value of the resistor. Show your work calculating resistance value.



(10) Question 6) Assembly array access. Write an assembly language subroutine called **CountIf** that reads all elements of an array. Each element of the array is a signed 8-bit integer. A pointer to the array is passed in R0, and the size of the array passed in R1. Count the number of elements that are less than or equal to 42. The size may be 0. Follow AAPCS rules. Return the count in R0.

```
CountIf
```

(5) Question 7) Local variables. The subroutine **`mySub`** has no input or output parameters. The subroutine allocates one 32-bit local variable `j`, and uses R7 stack frame addressing to access the local variable. Fill in the binding for the local variable.

```
.equ j,  [          ]  //binding for 32-bit local variable
mySub: SUB  SP,SP,#4   //allocate j
       PUSH {R7}
       MOV  R7,SP      //frame pointer using R7
//---------start of body-------------------
       MOVS R0,#42
       STR  R0,[R7,#j] //set j = 42
;---------end of body---------------------
       POP  {R7}
       ADD  SP,SP,#4 //deallocate j
       BX   LR
```

(10) Problem 8) FSM STG. Draw the state transition graph for a Moore FSM. There are three inputs and one output. Consider the three inputs as a binary integer, *I*, from 0 to 7. The input will determine the state of the LED. Turn the LED on if there are an odd number of bits in *I* (e.g., *I* =4 has an odd number of ones, 100.) Turn the LED off if there are an even number of bits in *I* (e.g., *I* =5 has an even number of one, 101.) The inputs and LED output are both in positive logic. Each state has a name, an output, and 8 arrows to next states labeled 0,1,2,…7. Just show the graph, no software needed.

Do not worry about the initial state:

(15) Question 9) FIFO. You are asked to finish the FIFO implementation that can store up to 4 values.

Before starting this question consider the following invocation sequence. Only the last line should fail because the FIFO is empty. Determine what should be the return values d1 d2 d3?

```
int main(void){ char d1,d2,d3,d4;
  Fifo_Init();
  Fifo_Put(5);
  Fifo_Put(6);
  Fifo_Put(7);
  d1 = Fifo_Get();
  d2 = Fifo_Get();
  d3 = Fifo_Get();
  d4 = Fifo_Get(); // this should fail
```

You cannot change the C code for these variables and functions. You cannot add additional global or static variables. You can add local variables. You may assume we never put a 0 into this FIFO, so we can return a 0 from Get if the FIFO is empty.

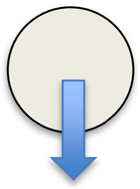
```
int32_t n;
char FIFO[4];

void Fifo_Init(void){
  n = 0; // empty
}
int Fifo_Put(char data){
  if(n == 4) return 0; // full
  FIFO[n] = data; // put data
  n++;
  return 1; // success
}
```

Hand execute the above main program and draw what is in the FIFO after the three puts. To test if your solution to Get is correct, hand execute the 4 calls to Get and see what is returned in d1 d2 d3 d4.

```
// Gets a character from the FIFO,
// Input: none
// Output: return the character or
//         return 0 if the FIFO was empty
char Fifo_Get(void){
```

(15) Problem 10) Sprites. Consider a game with many balls. Each ball has the following structure.

```
typedef enum {dead, dying, alive} status_t;
struct ball{
  int16_t x;        // x coordinate, in pixels
  int16_t y;        // y coordinate, in pixels
  int16_t vy;       // y velocity, in pixels/33.3ms
  int16_t ay;       // y acceleration, in pixels/33.3ms/33.3ms
  status_t life;};
typedef ball ball_t;
ball_t Balls[32];
int Flag; // 1 means Balls have moved and need to be redrawn
```



Implement a SysTick ISR that moves all alive balls. You may assume **ay** is constant, so balls move according to gravitational acceleration. On the other hand, first update **vy,** and then update **y** once every time the **ISR** is invoked. SysTick is triggered every 33.3ms. LCD output occurs in **Draw**, which is not shown and not asked for. **Draw** is called from the main program. If the **y** position of a ball goes above 159 or below 0, set its life to dying. Set the semaphore **Flag** as appropriate.

```
void SysTick_Handler(void){ // 30Hz periodic interrupt
```