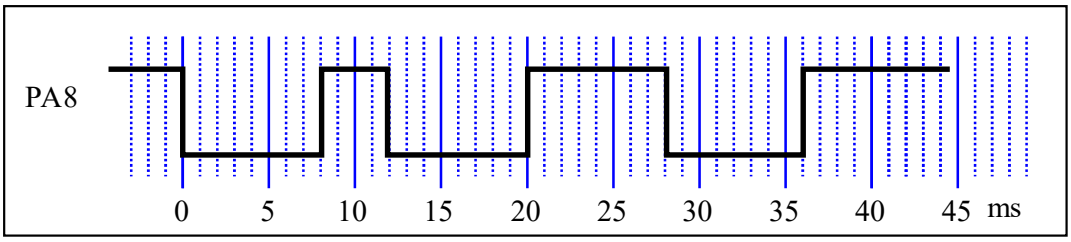
UT EID: _____ First: _____ Last: _____

Instructions:

- Closed book and closed notes. No books, no papers, no data sheets (other than the addendum)
- 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/blanks will be ignored in grading*. You may use the back of the sheets for scratch work.
- 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 and all subroutines AAPCS compliant.
- *Please read the entire exam before starting.*

(10) Question 1. Communications/UART. The baud rate is 250 bits/sec. The UART1 output uses PA8. The microcontroller outputs one frame with data equal to 0x32 (ASCII '2'). Draw the PA8 output signal as a function of time for this one frame. Assume the frame begins at time = 0. Assume the UART is idle before and after this one frame.



(10) Question 2. Friendly GPIO access. There is a 3-bit DAC connected to PB2-0. You can assume PB2-0 are initialized as outputs. Implement friendly functions that output to the DAC in both C and assembly. Assume input parameter is a value limited from 0 to 7.

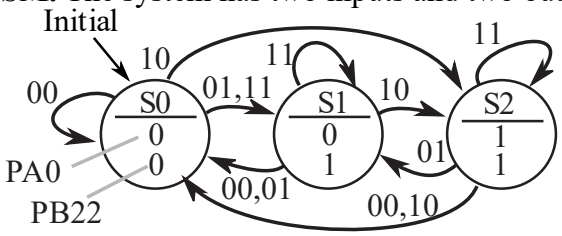
```
// R0 has 3-bit data
DAC3_Out:
  LDR R1,=GPIOB_DOUT31_0
  LDR R2,[R1]
  MOVS R3,#0x07
  BIC  R2,R2,R3
  ORRS R2,R2,R0
  STR  R2,[R1]
  BX LR
```

```
void DAC3_Out(uint32_t data){
uint32_t port;
  port = GPIOB->DOUT31_0;
  port = port&(~0x07);
  port = port|data;
  GPIOB->DOUT31_0 = port;
}
```

These will glitch/ click sound (hence bad)

```
  GPIOB->DOUTSET31_0 = data;
  GPIOB->DOUTCLR31_0 = data^7;
}
  GPIOB->DOUTCLR31_0 = 7;
  GPIOB->DOUTSET31_0 = data;
}
  GPIOB->DOUT31_0 &= ~7;
  GPIOB->DOUT31_0 |= data;
}
```

(10) Question 3. Moore FSM. The system has two inputs and two outputs.



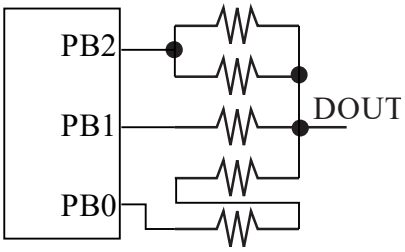
Part a) Write C code to define a struct for this FSM. Each state has an output value for PA0 and a separate output value for PB22. Each state also has four next states, but no time delay.

```
struct state{
  uint32_t OutPA0;
  uint32_t OutPB22;
  uint32_t Next[4];
};



typedef const state state_t; // do not change this line
```

Part b) Complete C code to define the state transition table in ROM. No engine is required.

```
state_t FSM[3]={
  {0,0,{S0,S1,S2,S1}},
  {0,1,{S0,S0,S2,S1}},
   {1,1,{S0,S1,S0,S2}}
};
```

(10) Question 4. DAC, Ohm's Law, KCL, KVL.



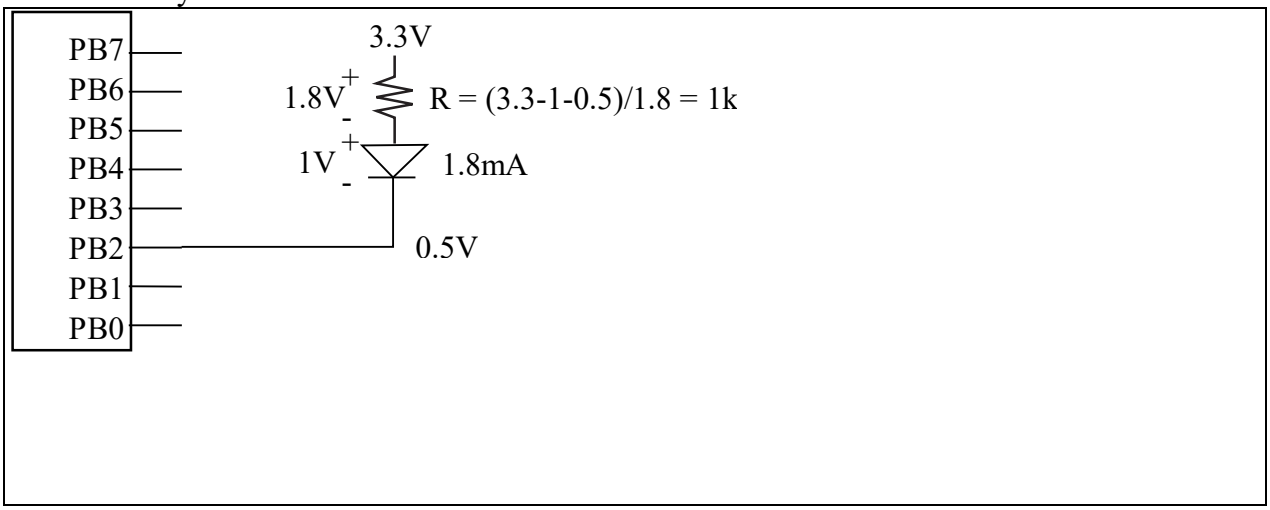
Hard way:
R2 = 5k, R1=10k, R0=20k, R1||R0=10*20/(10+20)=20/3k
DOUT = 7V*(20/3)/(5+20/3) = 7V*20/(15+20) = 4V

Easy way:
It's a binary-weighted DAC with range of 0 to 7V, precision of n=3 bits, so
resolution is $7V/(2^n-1) = 1V$, so DOUT = 4 V

(10) Question 5. LED. You are given this function, which should turn on an LED,

```
void LED_On(){
   GPIOB->DOUT31_0 &= ~0x04;
}
```

and are asked to interface an LED to the microcontroller, so the software operates as intended. The LED parameters are I_d = 1.8mA, V_d = 1V. The microcontroller output voltages are V_{OL}=0.5V and V_{OH} = 3.2V. Show the LED interface that makes this software work. Include math to determine any resistor values needed.



(10) Question 6) FIFO queue. There is exactly one line in **Get** that must be changed. Circle the line containing the bug in **Get**. Show the correction required so it operates correctly. Hint: execute two calls to **Put** and draw the resulting data structure. Then, execute two calls to **Get** to see if your correction fixes the bug.

```
uint32_t PutI;  // should be 0 to 7
uint32_t GetI;  // should be 0 to 7
int32_t static FIFO[8];
void Init(void){
  PutI = GetI = 0;
}
int Put(int32_t data){
  if(((PutI-1)&7) == GetI) return 0;
  FIFO[PutI] = data;
  PutI = (PutI-1)&7;
  return 1;
}
```

```
int Get(int32_t *datapt){

  if(PutI == GetI) return 0;

  *datapt = FIFO[GetI];

  GetI = (GetI+1)&7;   change + to -, GetI = (GetI-1)&7;

  return 1;

}
```

(10) Question 7. Local variables. The subroutine **mySub** has one call by value input parameter and one output parameter. The function must be AAPCS compliant. The C version is

```
uint32_t mySub(uint32_t x){ uint32_t z=10; return z*x);
```

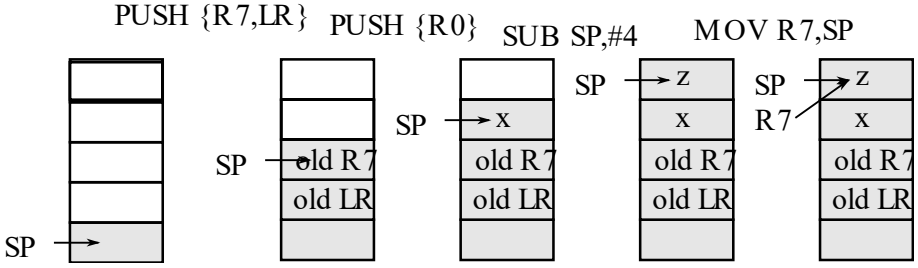
A typical calling sequence is

```
    LDR  R0,=1000                  uint32_t y;
    BL   mySub                     y = mySub(1000);
```

The input parameter **x** is passed in R0, but will be saved as a local on the stack. The subroutine allocates one 32-bit local variable, **z** and uses R7 frame pointer addressing to access the locals. The binding for these two locals are

```
.equ x,  [4]                  // binding for the input parameter x
.equ z,  [0]                  // binding for 32-bit local variable z
mySub: PUSH {R7,LR}
       PUSH {R0}              // parameter x is saved on the stack
       [SUB SP,SP,#4]         // allocate z
       MOV R7,SP              // establish frame pointer
//---------start of body-------------------
       MOVS R0,#10
       [STR R0,[R7,#z]]       // set z = 10, using R7
       LDR  R2,[R7,#x]  // R2 is input parameter x (1000)
       LDR  R3,[R7,#z]  // R3 is z (10)
       MULS R2,R2,R3    // R2 is z*x (10000)
//---------end of body---------------------
// balance the stack and return z*x
       [MOV R0,R2
       ADD SP,#8
       POP {R7,PC}]
```

Execute beginning instructions and then draw a stack figure

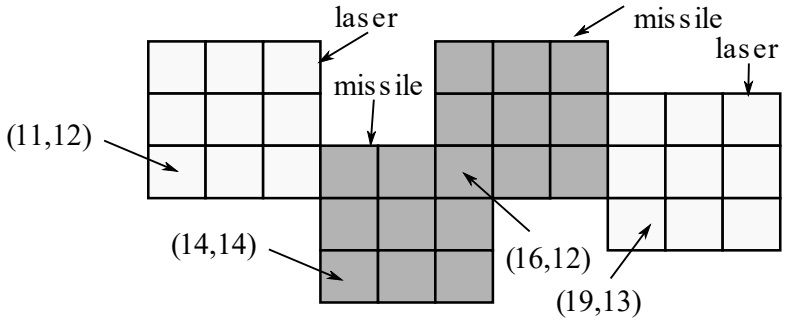


(15) Question 8. Arrays in assembly. Translate this C to assembly (assume I is initialized to 0)

```
uint16_t Buff[100];

uint32_t I;



void Dump(uint16_t x){
  if(I < 100){
    Buff[I] = x;
    I++;
  }
}
```

```
.data
// this is how to make globals in RAM
Buff: .space 200
I:    .space 4


.text
Dump:
// this is an "if" not a "loop"
 PUSH {R4,R5,LR}
 LDR  R2,=I      // pointer to I
 LDR  R3,[R2]    // value of I
 CMP  R3,#100
 BHS  skip       // full if I>=100
 LDR  R4,=Buff
 LSLS R5,R3,#1 // 2*I
 STRH R0,[R4,R5]
 ADDS R3,#1     // I+1
 STR  R3,R2     // I = I+1
skip:
 POP  {R4,R5,PC}
```

(15) Question 9. Collisions. Consider a game with 10 missiles and 20 lasers. There are two sprite arrays, **Missiles** and **Lasers**. Consider each sprite as a 3 by 3-pixel square. The (x,y) coordinate of a sprite is its lower left pixel. You may assume the arrays have been populated with data before your function is called. **Collision** is defined as the overlap of any pixel of a missile with any pixel of a laser. This first figure has no collisions.




```
typedef enum {dead,alive,dying} status_t;
struct sprite{
  int32_t x;         // x coordinate, in pixels
  int32_t y;         // y coordinate, in pixels
  int32_t vx;        // x velocity, in pixels/frame
  int32_t vy;        // y velocity, in pixels/frame
  status_t life;};   // dead or alive
typedef sprite sprite_t;
sprite_t Missiles[10];
sprite_t Lasers[20];
```

```
void Collisions(void){
 int32_t diff;
 for(int i=0; i<10; i++){ // missiles
  if(missiles[i].life == alive){
    for(int j=0; j<20; j++){ // lasers
      if(lasers[j].life == alive){
        diff = lasers[j].x-missiles[i].x;
        if((diff < 3)&&(diff > -3)){// -2,-1,0,1,2
          diff = lasers[j].y-missiles[i].y;
          if((diff < 3)&&(diff > -3)){// -2,-1,0,1,2
            missiles[i].life = dying;
            lasers[j].life = dying;
          }
        }
      }
    }
  }
}
int32_t dx.dy;
 for(int i=0; i<10; i++){ // missiles
  if(missiles[i].life == alive){
    for(int j=0; j<20; j++){ // lasers
      if(lasers[j].life == alive){
        dx = lasers[j].x-missiles[i].x;
        if(dx < 0) dx = -dx; // absolute value
        dy = lasers[j].y-missiles[i].y;
        if(dy < 0) dy = -dy; // absolute value
        if((dx < 3)&&(dy < 3)){// both are 0,1,2
          missiles[i].life = dying;
          lasers[j].life = dying;
        }
      }
    }
  }
}
int32_t dx,dy;
 for(int i=0; i<10; i++){ // missiles
  if(missiles[i].life == alive){
    for(int j=0; j<20; j++){ // lasers
      if(lasers[j].life == alive){
        dx = lasers[j].x-missiles[i].x;
        dy = lasers[j].y-missiles[i].y;
        if((dx*dx + dy*dy) < 9){// both are 0,1,2
          missiles[i].life = dying;
          lasers[j].life = dying;
        }
      }
    }
  }
}
```