

Net ID: _____ (no names, please)

You may use only the provided reference materials. You may use a calculator, either a four-function or a scientific calculator. You may not use a programmable calculator. The test is worth 100 pts, you are given 1 pt for free. For any required I2C functionality, use subroutine calls *i2c_start()*, *i2c_rstart()*, *i2c_stop*, *i2c_put(char byte)*, *char i2c_get(char ackbit)*. If you use *i2c_put*, you must pass in as an argument the byte that is to be written to the I2C bus. If you use *i2c_get*, you must pass in an as argument the bit value to be sent back as the acknowledge bit value. You also have *DelayUs()* and *DelayMs()* functions available. **Show all your work in any computations done or formulas used to receive full credit.**

Part I: (75 pts)

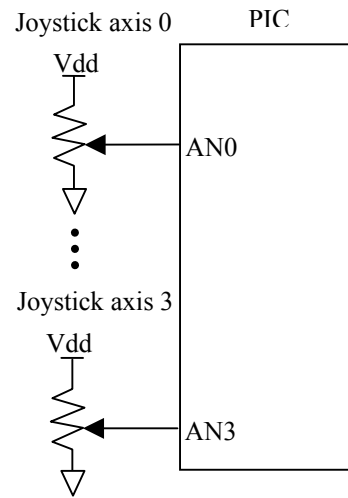
- a. (15 pts). The code fragment below performs I2C operations to a MAX517 DAC. Answer the questions in the box in the right (*Hint: Read the datasheet!!!!*)

```
// Operation 1
i2c_start();
i2c_put(0x5E);
i2c_put(0x10);
i2c_stop();

// Operation 2
i2c_start();
i2c_put(0x5E);
i2c_put(0x06);
i2c_put(0x5E);
i2c_stop();
```

1. What values are the A1, A0 lines on the MAX517 tied to (give as either VDD or GND for each)?
2. What do the instructions in operation 1 cause the DAC to do? If a voltage is output, give the voltage that appears on the DAC output assuming $V_{ref} = 5V$.
3. What do the instructions in operation 2 cause the DAC to do? If a voltage is output, give the voltage that appears on the DAC output assuming $V_{ref} = 5V$.

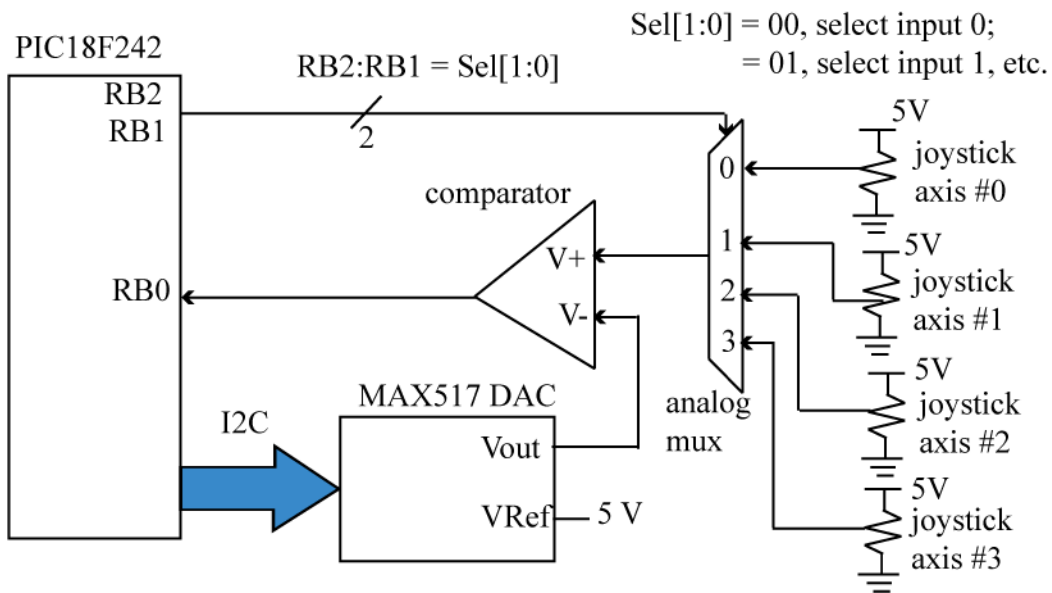
- b. (15 pts) A gamepad controller containing two analog joysticks is connected to the PIC. Each joystick has an X axis (left/right) and Y axis (up/down). Internally, each axis of the joystick is a potentiometer connected between power (Vdd) and ground; joystick position up or left gives Vdd while down or right gives ground. The center position produces Vdd/2. The joystick axes (four total, two for each joystick) are connected to AN0 through AN3. Write the C function `signed char whichDirection(unsigned char axis)` which reads a particular joystick axis specified by the `axis` parameter (values of 0, 1, 2, 3 correspond to reading analog inputs AN0, AN1, AN2, AN3 respectively). The function returns `UP_LEFT_DIR` when the joystick is at least halfway between the center position and full up or left, `DOWN_RIGHT_DIR` when the joystick is at least halfway between the center position and full down or right, and `CENTER_DIR` otherwise. Assume $V_{dd}=V_{ref+}$ is 5V, V_{ref-} is 0V and that the ADC is already configured with left justification. Use only the upper 8 bits of the resulting A/D conversion. You must delay for 20 μ s (use `DelayUs`) after selecting the A/D channel. Show all calculations, such as those used to convert voltages read from the joystick into `UP_LEFT_DIR`, `CENTER_DIR`, and `DOWN_RIGHT_DIR`, to receive full credit.



```
#define UP_LEFT_DIR 1
#define CENTER_DIR 0
#define DOWN_RIGHT_DIR -1

signed char whichDirection(unsigned char axis)
{
    // your code here
```

- c. (15 pts) A second method for reading a joystick is to compare the voltages using analog methods. Assume the joysticks work in the way as described in problem b. Write the C function `signed char whichDirectionAnalog(unsigned char axis)` which returns `UP_LEFT_DIR`, `DOWN_RIGHT_DIR`, or `CENTER_DIR`, depending on the joystick position, as was specified for problem b. As before, the axis parameter (0,1, 2, or 3) is used to select a joystick axis; in this case use this parameter to control the RB2, RB1 digital outputs to steer a joystick axis through the analog mux to the V+ input of the comparator as shown in the figure below. Using the MAX 517 DAC, output an analog voltage V- to compare against the joystick's current position (V+). The comparator output is '1' if $V+ > V-$; the comparator output is '0' if $V+ < V-$. Assume that A0 and A1 of the MAX 517 DAC are both tied high. Assume that the PORTB pins RB0, RB1, and RB2 have already been correctly configured (RB0 as an input, RB1, RB2 as outputs).



```
#define UP_LEFT_DIR 1
#define CENTER_DIR 0
#define DOWN_RIGHT_DIR -1
signed char whichDirection(unsigned char axis)
{
    // your code here
}
```

- d. (10 pts) Use the PWM module of the PIC18 to generate a square wave with a period of 15.2 μs and a high pulse width of 6.2 μs (the low pulse width is 9 μs). Show the calculations that you use to calculate the needed register values. Then write code for main() that configures the PWM for this operation. Your while(1){} loop should be empty. Use an Fosc value of 20 MHz. Assume the lower 2 bits of the 10-bit PWM pulse width value are already set to 0. Show all work.
- e. (10 pts) Write an ISR that responds to a CCPIF interrupt, which is being generated by the Timer1/CCP1 compare mode. On each CCPIF interrupt, increase the time until the occurrence of the next interrupt by 115.2 μs . The main() code which configures the compare module is given below, you need to look at this to discover how Timer1 is configured. Assume Fosc = 25 MHz. Show all work necessary to determine the timer tick rate and CCPR1 update value.

```
main()
{
    T1CKPS1 = 1; T1CKPS0 = 1; RD16 = 1; TMR1CS = 0;
    TMR1H = 0; TMR1L = 0;
    CCP1M3 = 1; CCP1M2 = 0; CCP1M1 = 1; CCP1M0 = 0;
    T3CCP2 = 0;
    CCP1IF = 0; CCP1IE = 1; IPEN = 0; PEIE = 1; GIE = 1;
    TMR1ON = 1;
    while (1);
}

void interrupt myIsr()
{
    // Your code here
}
```