

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 is 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)?

The I2C address is $0x5E = 0101 \underline{11}10$. The two underlined bits show **A1 = A0 = 1**.

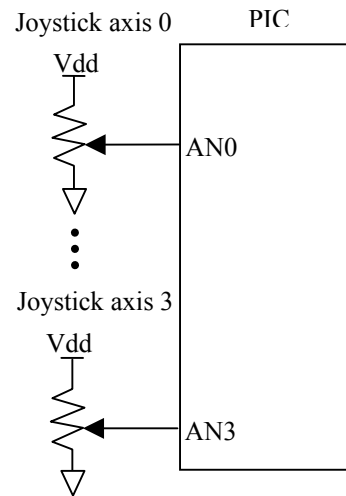
2. What do the instructions in operation 1 cause the DAC to do? If a voltage is output, give the voltage that appears on the DAC output assuming $V_{ref} = 5V$.

The command byte is $0x10 = 0001\ 0000$, which **resets all DAC registers**.

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 $V_{ref} = 5V$.

The command byte is $0x00 = 0000\ 0110$, which **outputs a voltage (bits 2,1 are don't cares)**. The voltage output is given by the next byte value sent, **0x5E**, and the **DAC voltage is $0x5E/256 * 5V = 94/256 * 5V = 1.84V$**

- 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



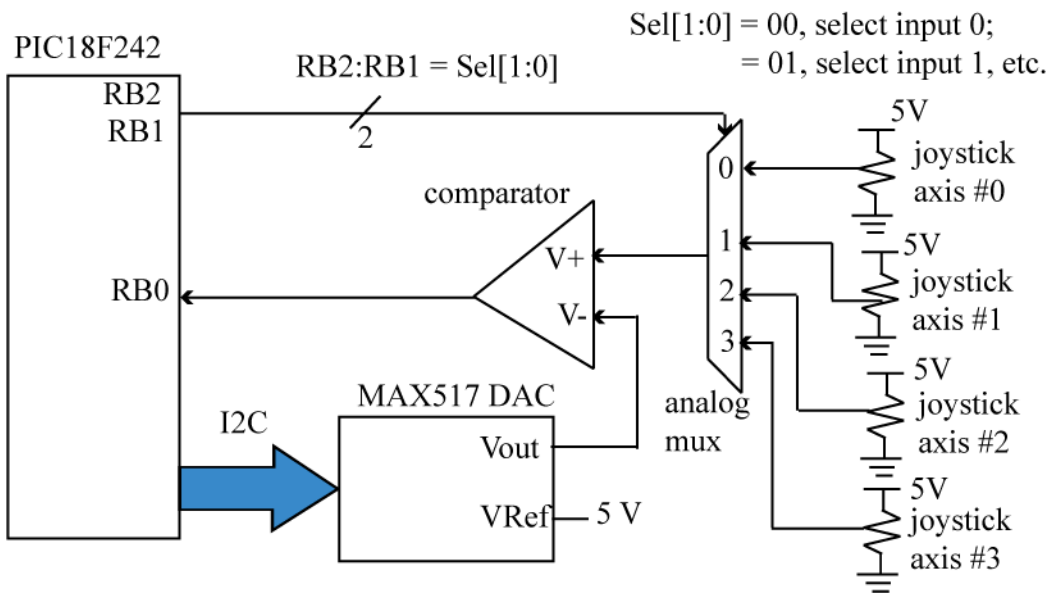
whichDirection(unsigned char axis) which reads a particular joystick axis specified by the axis parameter (values of 0, 1, 2, 3 correspond to reading analog inputs AN0, AN1, AN2, AN3 respectively). The function returns UP_LEFT_DIR when the joystick is at least halfway between the center position and 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 Vdd=Vref+ is 5V, Vref- 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 μs (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.

Note that UP_LEFT_DIR occurs when the input $> 3/4 V_{dd}$, while DOWN_RIGHT_DIR occurs when input $< 1/4 V_{dd}$. Since we're only using 8 bits, $3/4 V_{dd} = 3/4 * 2^8 = 192$ and $1/4 V_{dd} = 1/4 * 2^8 = 64$.

```
#define UP_LEFT_DIR 1
#define CENTER_DIR 0
#define DOWN_RIGHT_DIR -1
signed whichDirection(unsigned char axis){
    // your code here
    ADCON0 = (axis << 2) | 2; // Leave converter turned on,
                               // select channel for axis

    // another way that is less efficient:
    // switch (axis) {
    //     case 0: CHS2 = 0; CHS1 = 0; CHS0 = 0; break;
    //     case 1: CHS2 = 0; CHS1 = 0; CHS0 = 1; break;
    //     case 2: CHS2 = 0; CHS1 = 1; CHS0 = 0; break;
    //     default: CHS2 = 0; CHS1 = 1; CHS0 = 1; break;
    // }
    DelayUs(20); // Wait for A/D converter input to stabilize
    GODONE = 1; // Start conversion
    while (GODONE); // Wait for conversion to complete
    // Choose direction based on results
    if (ADRESH > 192) return UP_LEFT_DIR;
    if (ADRESH < 64) return DOWN_RIGHT_DIR;
    return CENTER_DIR;
}
```

- 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
    // Output 3/4 Vdd to the comparator
    switch (axis){
        case 0: RB2=0; RB1=0; break;
        case 1: RB2=0; RB1=1; break;
        case 2: RB2=1; RB1=0; break;
        default: RB2=1; RB1=1; break;
    }
    i2c_start();
    i2c_put(0x5F); // I2C address, write
    i2c_put(0); // Command - output D/A
    i2c_put(192); // Output 3/4 Vdd = 2^8 * 3/4
    i2c_stop();
    if (RB0) // If Vjoystick > 3/4 Vdd..

```

```
    return UP_LEFT_DIR;
// Output ¼ Vdd to the comparator
i2c_start();
i2c_put(0x5F); // I2C address, write
i2c_put(0);    // Command - output D/A
i2c_put(64);  // Output ¼ Vdd = 2^8 * ¼
i2c_stop();
if (!RB0)     // If Vjoystick < ¼ Vdd
    return DOWN_LEFT_DIR;
return CENTER_DIR;
}
```

- 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.

Choosing a 1:1 prescale and PR2 = 75, gives PWM period = $(PR2 + 1) * 4 * T_{osc} * PRE = (75 + 1) * 4 / 20 \text{ MHz} * 1 = 15.2 \mu\text{s}$.

Choosing a CCPR1L = 31 or 0001 1111 in binary and assuming the lower 2 bits of the 10-bit value are 0 gives CCPR1L:CCP1CON[5:4] = 00 0111 1100 = 124 decimal. Therefore, the PWM duty cycle is $(CCPR1L:CCP1CON[5:4]) * T_{osc} * PRE = 124 / 20 \text{ MHz} * 1 = 6.2 \mu\text{s}$.

```
main(){
    TMR2ON = 1;           // Turn timer2 on
    T2CKPS1 = 0; T2CKPS0 = 0; // Choose a 1:1 prescale
    PR2 = 75;            // Set PWM period to 15.2 us.
    CCP1M3 = 1; CCP1M2 = 1; // Choose PWM mode
    CCPR1L = 31;        // Choose a duty cycle of 6.2  $\mu\text{s}$ .
    TRISC2 = 0;         // Make PWM pin an output
    while (1);          // Run PWM
}
```

- 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(){
    // Timer 1 configured for divide by 8, so clock period is  $(4*8)/25 = 1.28 \mu\text{s}$ 
    // Select interrupt on compare mode, enable interrupts.
    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
    if (CCP1IF && CCP1IE){
        CCP1IF = 0; // Clear interrupt
        CCPR1 += 90; // 1.28 us / tick * 90 ticks = 115.2 us
    }
}
```

- f. (10 pts) Write C code which reads two bytes starting at location 0x7A40 from the 24LC515 EEPROM, where the 24LC515 has A1 = 1 and A0 = 1. The 2nd byte read from EEPROM must be sent to one of two different MAX517 DACs connected to the I2C bus. If the first byte is zero, write the 2nd byte to a MAX 517 which has its A1,A0 pins as A0 = 0 and A1 = 1; if the first byte is not zero, write the 2nd byte to a MAX 517 where A0 = 1 and A1 = 0. Assume that the EEPROM is ready for a read operation, so that no polling is necessary.

```

char first, second;
// Set the address to 0x7A34
i2c_start();
i2c_put(0xA6);           // I2C address: B0 = 0, A1 = A0 = 1, R/W = 0
i2c_put(0x7A);          // Upper byte of address
i2c_put(0x40);          // Lower byte of address
i2c_rstart();           // Switch to a read cycle
// Perform read
i2c_put(0xA7);          // I2C address: B0 = 0, A1 = A0 = 1, R/W = 1
first = i2c_get(0);     // Acknowledge first read
second = i2c_get(1);   // Don't acknowledge second read
i2c_stop();
// Decide where to write
i2c_start();
if (first) i2c_put(0x5A); else i2c_put(0x5C);
i2c_put(0x00);          // Command byte: send D/A value
i2c_put(second);
i2c_stop();             // Finish up

```

Part II: (24 pts) Answer 6 out of the next 8 questions. Cross out the 2 questions that you do not want graded. Each question is worth 4 pts.

1. When $F_{osc} = 40$ MHz and a 1:2 prescale is chosen for timer1, how much time occurs between TMR1IF interrupts? How can this time be increased?

$$\frac{1}{40 \text{ MHz}} \cdot \frac{8}{1} \cdot 2^{16} = \mathbf{13.1 \text{ ms.}}$$

(8 = PRE * 4) To increase, set the prescale higher.

2. What is the maximum possible PWM **frequency**? What is the minimum possible PWM **frequency**? Assume $F_{osc} = 10$ MHz.

Choosing a 1:1 prescale and setting $PR2 = 0$, the PWM period = $(PR2 + 1) * 4 * T_{osc} * PRE = (0 + 1) * 4 / 10 \text{ MHz} * 1 = 0.4 \text{ us}$ or **2.5 MHz**. Choosing a 1:16 prescale and $PR2 = 255$, the PWM period = $(1 + 255) * 4 / 10 \text{ MHz} * 16 = 16.4 \text{ ms}$ or **610 Hz**.

3. Recalling that the maximum clock period of the A/D converter is 1.6 us and given an F_{osc} of 1 MHz, what setting of the A/D conversion clock select bits in the ADCON2 register gives the highest possible sampling frequency? What is that frequency?

Sampling frequency = F_{osc} / scaling. Max permissible sampling frequency is $1 / 1.6 \text{ us} = 625 \text{ KHz}$. Choosing $F_{osc} / 2$ by setting **ADCS2:0 = 000** gives the highest possible sampling frequency of **500 KHz**.

4. On a single I2C bus, what is the maximum number of MAX 517 DACs that can be uniquely addressed? What is the maximum number of 24LC515 serial EEPROMS that can be addressed?

Because both the MAX 517 and the 24LC515 have two address pins, a maximum of $2^2 = 4$ of each chip can be addressed.

5. Assume the PWM module has been configured with $PR2 = 128$, and $CCPR1L = 130$. What would you expect to see on the output of the CCP1 pin? Explain why.

Because $CCPR1L$ is greater than $PR2$, the CCP1 pin will always stay high (a 100% duty cycle).

6. If a 10 MHz clock source is provided to a 12 bit flash A/D converter, what is its maximum sampling frequency (i.e., the number of conversions done in one second)? If a 10 MHz clock source is provided to a 12 bit successive-approximation A/D converter, what is its maximum sampling frequency?

A flash A/D converter takes 1 clock cycle to complete a conversion, so its maximum sampling frequency is **10 MHz**. A successive-approximation A/D converter takes 1 clock cycle per bit, so its maximum sampling frequency is $10 \text{ MHz} / 12 = \mathbf{833 \text{ KHz}}$.

7. How many bit times are required to read one byte from the 25LC515 EEPROM and write that byte to the MAX 517 A/D converter? Assume the EEPROM has completed all writes and count a start or stop as one bit time. Assume the desired read address to the 25LC515 has already been set, so do not include choosing a read address in the calculations.

EEPROM read: start + (I2C address + ACK) + (byte read + NACK) + stop = 1 + 9 + 9 + 1 = 20 bits

MAX 517 write: start + (I2C address + ACK) + (command byte + ACK) + (byte written + ACK) + stop = 1 + 9 + 9 + 9 + 1 = 29

Total: 20 + 29 = **49 bit times**.

8. If the cost of placing one resistor on a chip is 0.01 cents and the cost of a comparator is also 0.01 cents, how many bits of precision (1-bit? 2-bit? 3-bit? 100-bit?, etc) can a flash A/D converter possess with a budget of \$1 per chip?

Recalling that there are 2^n resistors and $2^n - 1$ comparators in an n-bit flash A/D converter, then there are approximately $2 \cdot 2^n = 2^{n+1}$ such components in an n-bit chip. At 0.01 cents per, \$1 gives us 10,000 components. Recalling that $8K = 8096 = 2^3 * 2^{10}$, we can buy a 12 bit chip.