

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.

Part I: (75 pts)

- a. (15 pts). The code fragment below performs I2C operations to a 24LC515 serial EEPROM. Answer the questions in the box in the right.

```
char buf[3];
```

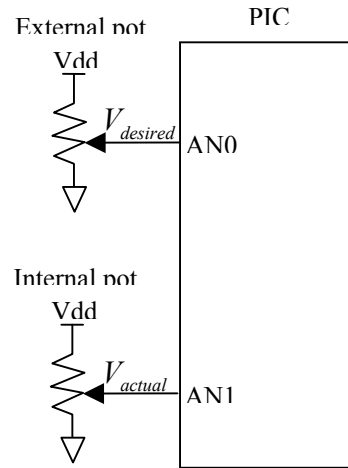
```
i2c_start();  
i2c_put(0xAE);  
i2c_put(0xEA);  
i2c_put(0xAE);  
i2c_restart();  
i2c_put(0xAF);  
buf[0]=i2c_get(0);  
buf[1]=i2c_get(0);  
buf[2]=i2c_get(1);  
i2c_stop();  
i2c_start();  
i2c_put(0xA6);  
i2c_put(0x7A);  
i2c_put(0xA7);  
i2c_put(buf[2]);  
i2c_put(buf[1]);  
i2c_put(0xB1);  
i2c_stop();
```

1. What values are the A1, A0 lines on the EEPROM tied to (give as either VDD or GND for each)?

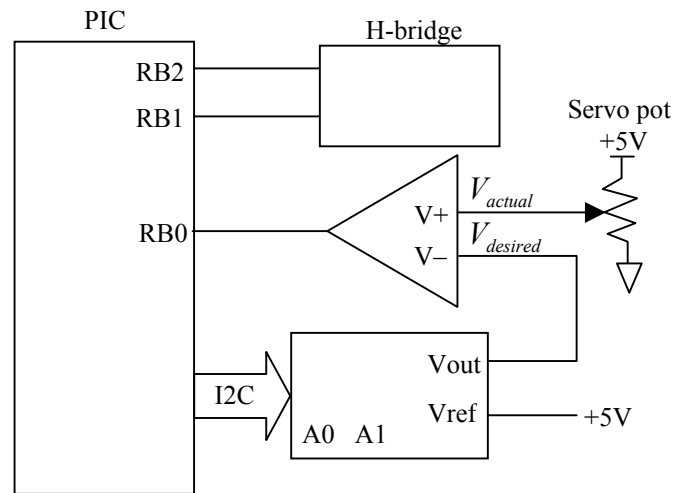
2. What locations in the SERIAL EEPROM are read? Give answers in hex.

3. What locations in the SERIAL EEPROM are modified? Your answer(s) must be in the form of either “The value 0x?? is written to location 0x????”, or “The content of location 0x???? is transferred to location 0x????”. In each case, ‘location 0x????’ is a location in the EEPROM. Give all values in hex.

- b. (15 pts) Given the setup shown to the right in which an external potentiometer connected to AN0 specifies the desired servo position as the voltage $V_{desired}$ (as in lab 11) and a second potentiometer inside the servo connected to AN1 specifies the actual servo position as a voltage V_{actual} (unlike lab 11), write the C function `signed char servoVoltage()` to determine which direction the servo's motor should spin in order to move from the current position to the desired position. If $|V_{desired} - V_{actual}| < 0.25V$, return 0, indicating the motor is close enough and should not turn in either direction. Otherwise, return 1 when $V_{desired} > V_{actual}$ and -1 when $V_{desired} < V_{actual}$ to turn the motor in the appropriate direction. Assume V_{ref+} is 5V, V_{ref-} is 0V and that the ADC is already configured with left justification. You must delay for $20\ \mu s$ after selecting the channel using `DelayUs`. You must show the calculations you used to convert the values read from the ADC into voltages which can be used in the formula above.



- c. (15 pts) A second method for controlling a servo is to compare the voltages using analog methods then drive a H-bridge based on the results of the comparison. Given a desired angle in degrees, which ranges from 0° to 180° , output an analog voltage $V_{desired}$ which ranges from 0V to +5V using the I2C bus to control a MAX 517 DAC, where 0V corresponds to 0° and +5V corresponds to 180° . The comparator will compare this voltage with the voltage V_{actual} read from the potentiometer inside the servo, which specifies the servo's current position. Using the result of the comparison, input through RB0, to choose the motor direction by outputting values on RB1 and RB2. When $V_{desired} < V_{actual}$, output RB1 = 1 and RB2 = 0; when $V_{desired} > V_{actual}$, output RB1 = 0 and RB2 = 1. Write a C function `void servoControl(unsigned char degrees)` which performs this function. Use the I2C functions to communicate with the MAX517 DAC. 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.



- d. (10 pts) Use the PWM module of the PIC18 to generate a square wave with a period of 200 μs and a high pulse width of 50 μs (the low pulse width is 150 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. Do not worry about the lower 2-bits of the 10-bit PWM pulse width value.
- e. (10 pts) Assume that the CCP1 module is configured to generate a compare match 30 ms after TMR1 is set to 0 by using compare mode. A low-true pushbutton is connected to RB0. Write an ISR that zeros TMR1 when the button is first pressed, enables the CCP1IF interrupt, and then ignores button press interrupts until 30 ms has expired. After 30 ms, disable the CCP1 interrupt and re-enable the button press interrupt, and set the semaphore variable `BUTTON_PRESSED` to a '1'. Assume `main()` code already exists which properly configures CCP1, TMR1, and INT0 (falling edge triggered).

