You may use a calculator and the provided reference materials. If a binary result is required, give the value in HEX. For any required I2C functionality, use subroutine calls to 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 as an argument the bit value to be sent back as the acknowledge bit value.

Part I: (76 pts) You must answer all of these questions.

a. (10 pts) Write a function called char adc_chk (char chan) that performs a PIC18 conversion on ADC input chan and returns a ‘1’ if
2.0 V \leq \text{adc\_val} \leq 3.0 V, where \text{adc\_val} is the upper 8-bits of the 10-bit PIC18 ADC value. The function returns a ‘0’ is \text{adc\_val} is out of this range. Assume Vref+ is 5 V, Vref- is 0V and that the ADC is already configured, with left justification. The \text{chan} value can be 0 to 3, with ‘0’ selecting AN0, ‘1’ selecting AN1, ‘2’ selecting AN2, and ‘3’ selecting AN3. You must delay for 20 µs after selecting the channel using DelayUs.

Translate 2V, 3 V into binary values returned by the upper 8-bits of the PIC18 ADC:
2 V/ 5 V * 256 = 102 (low bound); 3V/5V * 256 = 154 (high bound)

```c
char adc_chk(char chan) {
    switch(chan) {
    case 0: CHS2=0;CHS1=0;CHS0 =0; break; //select AN0
    case 1: CHS2=0;CHS1=0;CHS0 =1; break; //select AN1
    case 2: CHS2=0;CHS1=1;CHS0 =0; break; //select AN2
    default: CHS2=0;CHS1=1;CHS0 =1; break; //select AN3
    }
    DelayUs(20);
    GODONE = 1;     //start conversion
    while(GODONE); //wait for conversion to finish
    if (ADRESH >= 102 && ADRESH <= 154) return(1);
    else return(0);
}
```
b. (10 pts) Write a loop that causes the MAX 517 to output a voltage of VREF (or as close as you can get), and decrease all of the way to 0 V in steps of 1 LSb (least significant bit). You must use I2C functions when causing the MAX517 to perform a conversion.

```c
char  dac_value;
dac_value = 0xFF;  // initial value is as close to VREF  
                   // as possible

do {
    i2c_start();
    i2c_put(0x58);   //dac address byte, assume A1=A0=0
    i2c_put(0x00);   //dac command byte
    i2c_put(dac_value);  //value to output
    i2c_stop();
    dac_value--;         //decrement to next voltage
} while(dac_value != 0xFF);  //exit when we wrap around back  
                           // to 0xFF
```

c. (10 pts) Write a C code fragment that reads three bytes starting at location 0xBF08 in the 24LC515 serial EEPROM into char variables a, b, and c. You must use the I2C function calls listed at the start of the test. Assume the A1, A0 lines of the EEPROM are tied high. When you begin reading bytes, you must read them in the same I2C transaction.

```c
char  a,b,c;

i2c_start();      //do write to set address
i2c_put(0xAE);    //addr byte 1010 1 1 1 0, LSb=0 for write
i2c_put(0xBF);    //MSB address
i2c_put(0x08);    //LSB address
i2c_restart();    //abort write, start read
i2c_put(0xAF);    // addr byte 1010 1 1 1 1, LSb=1 for read
a = i2c_get(0);   // get byte, send ACK
b = i2c_get(0);   // get byte, send ACK
C = i2c_get(1);   // get byte, send NAK
i2c_stop();
```
d. (10 pts) Use the PWM module of the PIC18 to generate a square wave with a period of 500 µs and a high pulse width of 100 µs (the low pulse width is 400 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 10 MHz.

\[ 500 \text{ us} = (\text{PR2}+1) \frac{4}{10 \text{ Mhz} \times \text{PRE}} \]

\[
\text{PR2} = \left[ \frac{500 \text{ us} \times 10 \text{ MHz}}{4 \times \text{PRE}} \right] - 1;
\]

for PRE=1, PR2 = 1249 (too large)
for PRE=4, PR2 = 311 (too large)
for PRE=16, PR2 = 77    use this

Duty cycle = 100/500 = 0.2 (20%)

\[ \text{CCPR1L} = 0.2 \times (\text{PR2}+1) = 15.6 = 16 \]

```
main()
{
    PR2 = 77;  CCPR1L = 16; TRISC2 = 0;
    T2CKPS1 = 1; //psecale 16
    CCP1M3 = 1; CCP1M2 = 1; //pwm mode
    TMR2ON = 1;
    while(1);
}
```

e. (6 pts) How many Timer1 tics are in 5 ms assuming an FOSC = 16 MHz and a prescale of 8?

\[
\text{time} = \#\text{of Tics} \times \text{timer clock period} = \#\text{of Tics} \times \frac{4}{\text{FOSC} \times \text{PRE}}
\]

\[
0.005 = \#\text{tics} \times \left( \frac{4}{16 \text{ MHz} \times 8} \right)
\]

\[
\#\text{tics} = \left( \frac{0.005 \times 16 \text{ MHz}}{4 \times 8} \right) = 2500
\]

f. (6 pts) When we measured pulse width from a pushbutton switch using capture mode, pin CCP1, and Timer1, we used two interrupt flags in the ISR. What were these interrupt flags and what was each one used for?

- **TMR1IF** - used to count the number of timer overflows between the falling edge (press) and the rising edge (release).
- **CCP1IF** - used to capture the timer1 value in the CCP1 register on the falling edge (press), then used to capture the timer1 value in the CCP1 register on the rising edge (release).
g. (8 pts) Assume the code used in lab to measure the pulse width of a pushbutton switch. On the falling edge (pushbutton pressed), the capture register captures the hex value 0xFFF0 from timer1. On the rising edge (pushbutton released), the capture register captures the value 0x1000 from Timer1, with THREE timer1 overflows between the falling and rising edge captures. Give the answer in decimal.

\[
#\text{tics} = (\#\text{offlows} - 1) \times 65536 + (0x0000 - \text{first capture}) + \text{second capture}
\]

\[
= 2 \times 65536 + (0x0000 - 0xFFF0) + 0x1000 = 131072 + 0x0010 + 0x1000
\]

\[
= 131072 + 16 + 4096 = 135184
\]

h. (10 pts) Explain EITHER the operation of a 3-bit successive approximation ADC or a 3-bit flash ADC. For both ADCs, use Vin = 3.2 V and Vref = 4 V. If you explain the successive approximation ADC, you have to give the internal VDAC voltage used at each comparison step, and list all steps. If you explain a flash ADC, you have to give the number of comparators and resistors, the output value (1 or 0) of all comparators. For either ADC, you have to give the final 3-bit output code.

**Successive Approximation:**
First guess: 100 , produces voltage 4/8*4V= 2V. This is less than Vin (3.2V), so first bit is 1
2nd guess: 110 produces voltage 6/8*4V = 3V. This is less than Vin(3.2V), so 2nd bit is 1
3rd guess 111 produces voltage 7/8 * 4V = 3.5V. This is greater than Vin (3.2V), so last bit is 0.
Final answer: 110.

**Flash:** 7 comparators, 8 resistors. Each resistor represents 4V/8 = 0.5 steps. The comparator input voltages are (from top comparator to bottom comparator): 3.5 V, 3.0, 2.5, 2.0, 1.5, 1.0,0.5.
Thus, the output of the comparators are 0 1 1 1 1 1 1, with the 3 bit output being 110 (6) since the output of six comparators are equal to ‘1’.

i. (6 pts) In lab, a sine wave was generated from a 64-entry lookup table using a periodic interrupt. If we wanted the sinewave to have a frequency of 50 Hz, then what does the interrupt interval have to be? Give the answer in microseconds.

\[
\text{sinewave period} = 1/50 \text{ Hz} = 0.02 \text{ seconds}
\]

\[
64 \text{ table entries} \times \text{interrupt interval} = 1 \text{ sinewave period}
\]

\[
\text{interrupt interval} = \text{period}/64 = 0.02/64 = 312.5 \text{ us}
\]
(5 pts) Part II: (24 pts) Answer 8 out of the next 10 questions. Cross out the 2 questions that you do not want graded. Each question is worth 3 pts, there is no partial credit.

1. Write C code that configures the Capture Compare Module to capture a rising edge on the CCP1 pin with a prescale of 1, and use the TIMER3 as the timebase. You DO NOT have to configure TIMER3.

```c
CCP1M3 = 0; CCP1M2=1; CCP1M1 = 0; CCP1M0 =1; //capture mode
T3CCP2 = 1;  //select timer 1 as a time base
```

2. Given an FOSC = 6 MHz, what is the fastest internal clock choice other than the internal A/D oscillator that can be used to drive the ADC clock and not violate the clock period constraint of 1.6 μs?

Max clock freq = 1/ 1.6 us = 625 KHz
6MHz/8 = 750 KHz, too fast
6 MHz/16 = 375 KHz, this is fastest that works.

3. Write a C code fragment that returns the upper 8-bits of the PIC18 ADC result value in the char variable `c` regardless of whether the ADC is configured as left justified or right justified.

```c
char adc_val;

if (ADFM == 0) {
    c = ADRESH;     //left justified
} else {
    //right justified, combine lower 2 bits of ADRESH
    // with upper 6 bits of ADRESL
    c = (ADRESH << 6) | (ADRESL >> 2);
}
```
4. Write a C code fragment that will initiate a STOP condition on the I2C bus and wait for it to be finished using the PIC18. (do not use the `i2c_stop` function, I want to know what is inside the `i2c_stop` function!)

```c
PEN = 1;
while(PEN); //wait for start to finish
```

5. How many bit times are there in the I2C transaction to the MAX517 DAC for a conversion? Count the start and stop conditions each as one bit time.

1 start+ 1 stop + addr byte (9) + cmd byte(9) + data byte(9)
29 bit times total

6. Assume the PIC18 A/D is configured with a Vref+ = 4V, and a Vref- = 0 V, and that two successive reads of the ADC are done, with the first value returning 0xA0, and the next value returning 0x80 (only the most significant 8 bits are read). What is the change in voltage?

Difference is 0xA0 - 0x80 = 0x20 = 32.
32/256 * 4 V = 0.5 V change

7. In using PWM to control either voltage or current, what is varied? What is kept constant?

Period is kept constant, pulse-width (duty cycle) is varied.
8. In space-width encoding, what is used to distinguish a ‘1’ from a ‘0’? Rising or falling edge transition? Period? Duty-Cycle?

The period is used to distinguish a ‘1’ from a ‘0’ (duty cycles may be different also, but it is the period that matters).

9. Assume that I calculate a pulse-width on the INT0 input by generating an interrupt on the falling edge, read the Timer0 value, then generate an interrupt the rising edge, read the Timer0 value, and subtract the two values. What is the inaccuracy with this measurement? (do not worry about timer overflow, that is not the issue here).

The amount of time from the first edge occurring to the timer being started within the ISR is not counted, and the amount of time from the second edge occurring and the timer being read is erroneously counted as part of the pulse width.

10. When it is necessary for the PIC18 to provide an ACK bit value of ‘1’ (a NAK) to the 24LC515 serial EEPROM?

During a read of the EEPROM, a NAK (ack bit = ‘1’) is sent for the last byte that is read from the EEPROM; ACKs (ack bit = ‘0’) are sent for the other bytes.