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: (68 pts) You must answer all of these questions.

a. (14 pts) Write a function called `char adc_chk (char chan)` that performs a PIC18 conversion on ADC input `chan` and returns a ‘1’ if $2.0 \, \text{V} \leq \text{adc\_val} \leq 3.0 \, \text{V}$, where `adc\_val` is the upper 8-bits of the 10-bit PIC18 ADC value. The function returns a ‘0’ if `adc\_val` is out of this range. Assume `V_{ref+}` is 5 V, `V_{ref-}` is 0V and that the ADC is already configured, with left justification. The `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`.

```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);
}
```

Translate 2V, 3 V into binary values returned by the upper 8-bits of the PIC18 ADC:
$2 \, \text{V} / 5 \, \text{V} \times 256 = 102$ (low bound); $3 \, \text{V} / 5 \, \text{V} \times 256 = 154$ (high bound)
b. (14 pts) For the hardware setup shown below, write a C function that determines if
voltage $V_a$ is greater than, less than, or equal to the voltage $V_b$ (use a ‘printf’ statement
with a message to this effect). One approach is to start the MAX 517 at 0 V, then increase
this voltage in a loop until one (or both) of the comparator outputs becomes a ‘0’ value –
this will tell you the relationship between voltages $V_a$ and $V_b$. You could also start the
MAX 517 at 5 V, and decrease this voltage in a loop until one or both of the comparator
outputs becomes a ‘1’. Assume that the ports RB1 and RB0 are already configured before
entering the function.

Start DAC voltage at 0V, and increase it by 1 LSB each time through the loop. Both RB1 and RB0 start at
‘1’, because both $V_a$ and $V_b$ will be greater than the DAC voltage.
If RB0 becomes ‘0’ first, then $V_b < \text{DAC voltage}$, and $V_a > \text{DAC voltage}$, so $V_a > V_b$.
If RB1 becomes ‘0’ first, then $V_a < \text{DAC voltage}$, and $V_b > \text{DAC voltage}$, so $V_b > V_a$
If RB0 and RB1 simultaneously become ‘0’, then $V_a = V_b$, and $V_a, V_b < \text{DAC voltage}$.

```c
void my_solution(void) {
    char dac_value;
    dac_value = 0;    //start DAC output at 0V
    do {
        i2c_start();
        i2c_put(0x5E);  //dac i2c address
        i2c_put(0x00);  //command byte to do conversion
        i2c_put(dac_value);  //value to converter
        i2c_stop();
        DelayUs(10);    //give DAC time to settle
        dac_value++;
    }while (RB0 && RB1);  //loop until at least one becomes ‘0’
    if (!RB0 && !RB1) printf("Va is equal to Vb");
    else if (!RB0) printf("Va is greater than Vb");
    else printf("Va is less than Vb");
}
```
c. (14 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.

```
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. (13 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.

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

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

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

\[
\text{Duty cycle} = \frac{100}{500} = 0.2 \text{ (20%)}
\]

\[
\text{CCPR1L} = 0.2 \times (\text{PR2}+1) = 0.2 \times 78 = 15.6 = 16
\]

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

e. (13 pts) Explain EITHER the operation of a 3-bit successive approximation ADC or a 3-bit flash ADC. For both ADCs, use Vin = 2.8 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 (2.8V), so first bit is 1
2nd guess: 110 produces voltage 6/8*4V = 3V. This is greater than Vin(2.8V), so 2nd bit is 0
3rd guess 111 produces voltage 5/8 * 4V = 2.5V. This is less than Vin (2.8V), so last bit is 1.
Final answer: 101.

**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 0 1 1 1 1 1 ,
with the 3 bit output being 101 (5) since the output of five comparators are equal to ‘1’.
Part II: (32 pts) Answer 8 out of the next 10 questions. Cross out the 2 questions that you do not want graded. Each question is worth 4 pts.

1. Draw a diagram that shows a PIC microcontroller connected to two different MAX517 DACs. Label all of the pins.

![Diagram of PIC microcontroller connected to two MAX517 DACs]

SCL: Clock
SDA: Data
Both SCL, SDA are bidirectional

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. If the Vref of the PIC18 is 4.1 V, and I read a 10-bit code of 0x200, what is the input voltage value?

\[
0x200 = 512, \quad 2^{10} = 1024. \quad \frac{512}{1024} \times 4.1 = 2.05 \text{ V}
\]

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. Write PIC18 code that configures TIMER2 for prescale of 4, and postscale of 11, internal clock as the clock source, and turns the timer on.

\[
\text{T2CON} = 0x55; \quad // \quad 01010 1 01
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad // \quad \text{post} \quad \text{tmron} \quad \text{pre}
\]

7. What does the least significant bit of the first byte of every I2C transaction signify?

If it is a read or write transaction; ‘1’ for read, ‘0’ for write.
8. Given a 9 bit DAC, and a reference voltage of 4.096 V, what is the expected output voltage change on the DAC output if the input code is changed by 1 Least Significant Bit? Give the answer in millivolts.

\[
\frac{1}{2^9} \times 4.096 = \frac{1}{512} \times 4.096 = 0.008 = 8 \text{ mV}
\]

9. In class and in the notes, we discussed the proper way to WRITE a 16-bit value to timer1 so that it is updated correctly. Show C code that updates TIMER1 with the value from ‘int my_value’ correctly.

```
TMR1H = (char) my_value/256; //or my_value >> 8;
TMR1L = (char) my_value & 0xFF; //mask off upper 8 bits
```

10. How many Timer1 tics are in 25 ms assuming an FOSC = 16 MHz and a prescale of 8?

```
time = ticks * timer_clock_period  
ticks = time/ timer_clock_period

\text{Timer clock period} = \frac{4}{\text{FOSC} \times \text{PRE}}  
= \left(\frac{4}{16,000,000}\right) \times 8 \approx 0.000002 = 2 \text{ us}
```

\[
ticks = \frac{time}{\text{timer_clock_period}} = \frac{0.025}{0.000002} = 12,500 \text{ ticks}
\]