

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)*, *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. (15 pts) Assume an FOSC of 10 MHz. Write a **main()** function that will use PWM mode to generate a square wave with a 75% duty cycle and a period of 6 KHz on the CCP1 output pin. Divide your solution into two parts. In the first part, show any calculations required. In the second part, show the **main()** code. The last statement in the main code should just be an infinite **while(1){}** loop as the PWM hardware does everything for generating the square wave. The PWM duty-cycle is set by a 10-bit value, I only care about the upper 8-bits (assume the lower 2-bit are zero). Select the prescale that gives the LARGEST value for PR2 that still fits in the PR2 register.

(7 pts) Calculations here (show all work):

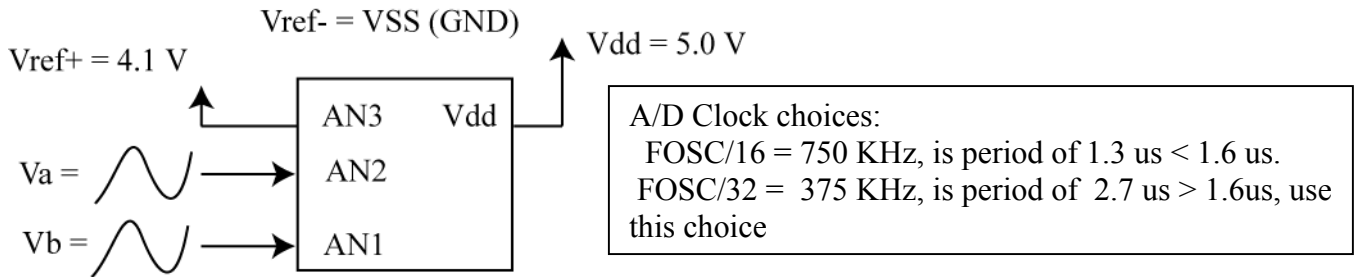
Timer2 PWM period = $(PR2+1) * (4/FOSC) * PRE$ (POST is NOT used for PWM period)
 $(1/6 \text{ KHz}) = (PR2+1) * (4/10 \text{ MHz}) * PRE$
 $PR2 = [(10 \text{ MHz}/4) / (6 \text{ KHz} * PRE)] - 1;$
 For PRE = 1, PR2 = 416 (> 255, too large)
 For PRE = 4, PR2 = 103
 For PRE = 16, PR2 = 25, so use PRE=4, PR2 = 103 = 0x67.
 For 75% duty cycle, CCPR1 = $0.75 * 103 \sim 77 = 0x4D$.

(8 pts) Code here:

```
main() {
    T2CKPS1 = 0; T2CKPS0 = 1; // pre scale of 4
    // configure PWM
    PR2 = 103; // set PR2 to max value
    CCPR1L = 77;
    //clear lower 2 bits of duty cycle
    bitclr(CCP1CON, 5);
    bitclr(CCP1CON, 4);
    bitclr(TRISC,2); // set CCP1 output

    // Set PWM Mode
    bitset(CCP1CON, 3); bitset(CCP1CON, 2);
    TMR2ON = 1;
    while(1); // infinite loop, PWM hardware does all work
}
```

- b. (15 pts) (PART 1, 7 pts) For the diagram below, write C code that will configure the A/D module for left justification, AN2, AN1 as analog inputs, AN3 as VREF+. VSS as VREF-. Use the internal FOSC clock, and configure the A/D clock such that it meets the minimum clock period constraint of 1.6 μ s assuming an FOSC of 12 MHz (use the fastest internal clock choice that meets this constraint). For the configuration code, use individual bit names ADCS2, ADCS1, ADCS0, ADON, ADFM, PCFG3, PCFG2, PCFG1, PCFG0. You do not have to configure the channel select bits, that is done in PART2

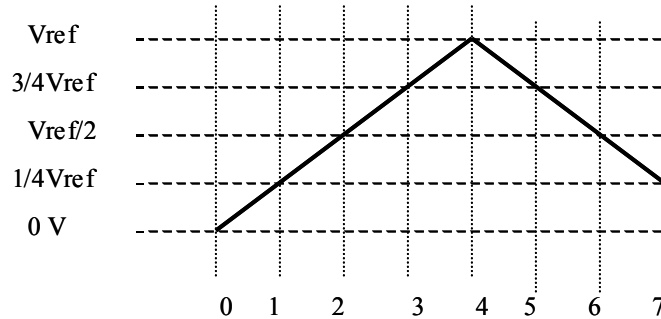


```
// AD Configuration, ADCON0 register
ADCS2 = 0; ADCS1 = 1; ADCS0 = 0; //FOSC/32
ADON = 1; //A/D turned on
ADFM = 0; // left justified
//AN2, AN1 analog inputs, AN3 = VREF, multiple solutions
PCFG3 = 0; PCFG2 = 0; PCFG1 = 0; PCFG0 = 1;
```

(PART 2, 8 pts) Write a function called 'char analog_sum()' that will perform a conversion on each analog input (AN2, AN1) and return the sum of these values as a 'char' value. Do not let the sum exceed 255 (0xFF) (hint: You will need to use an unsigned INT variable to hold the sum, then clip this to 255). When changing A/D channels, use the DelayUs function to delay 20 μ s to give the A/D input a chance to settle. Since you don't know how often this function will be called, use this delay before starting each conversion.

```
char analog_sum(){
    unsigned int tmp; char tmpc;
    CHS2=0;CHS1=0;CHS0=1; //select channel 1
    DelayUs(20); // wait for settling
    GODONE= 1; //godone bit = 1
    while (GODONE); // wait for finish
    tmp = ADRESH; // read result
    CHS2=0;CHS1=1;CHS0=0; //select channel 2
    DelayUs(20);
    GODONE= 1; //godone bit = 1
    while (GODONE); // wait for finish
    tmp = tmp + ADRESH; // add new result to old result
    if (tmp > 255) tmp = 255;
    tmpc = tmp;
    return(tmpc);
}
```

- c. (11 pts) We want to generate the waveform below using an 8-input lookup table. Assume a V_{ref} value of 5V, and an 8 bit DAC. The horizontal axis indicates the indices of the lookup table.



Fill in the contents of the 8 element char array used to

```
char my_table[] = { ??? } // 8 entries.
```

To make it easier to grade, fill in the table below with your 8 entries (the array indices are shown). Write your values in hex; this is the value to write to the DAC.

0: 0x00	4: 0xFF
1: 0x40	5: 0xC0
2: 0x80	6: 0x80
3: 0xC0	7: 0x40

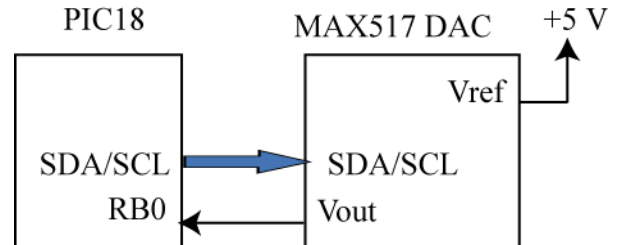
In general:

$$V_{in}/V_{ref} * 2^N = A/D \text{ output code}$$

For example, for $V_{in} = \frac{1}{4} V_{ref}$,

$$\frac{1}{4} V_{ref}/V_{ref} * 2^8 = \frac{1}{4} * 256 = 64 = 0x40$$

- d. (10 pts) In the diagram below, the output of the MAX517 D/A analog output is tied to the RB0 input. We would like to determine exactly what D/A output voltage is interpreted as a logic “1” on the RB0 input. Assume all configuration has been done (serial port, parallel ports, I2C, etc). Write a loop that starts the D/A output at 0 V and increments it each time through the loop by 1 LSB (one least significant bit value). Exit the loop when the RB0 input reads as a “1” value, and print the 8-bit DAC input code to the screen. You will need to use the I2C functions to write to the DAC. Assume both A1 and A0 on the DAC are tied high.



```

char dac_value;
dac_value = 0xFF; // 0xFF + 1 = 0x00, first byte to write to DAC

do {
    i2c_start();
    i2c_put(0x5E); // address byte
    i2c_put(0x00); // command byte
    dac_value++; // increment dac value
    i2c_put(dac_value); // data byte
    i2c_stop();
}
while (!RB0);
printf("DAC Value is %x", dac_value);
pcrlf();
  
```

- e. (10 pts) Write a sequence of I2C function calls that will return the byte from location 0x80F0 within the 24LC515 Serial EEPROM. Assume both A1 is tied low and A0 is tied high on the EEPROM. Write the byte read from the EEPROM into the variable *data_byte*.

```
i2c_start(); // 1 0 1 0 1 0 1 0, blk sel=1, R/W = 0
i2c_put(0xAA); // address byte, write command, high block
i2c_put(0x80); // high byte
i2c_put(0xF0); //low byte
i2c_rstart();
i2c_put(0xAB); // read command, 1010 1011, R/W = 1
data_byte = i2c_get(1); // read byte, ack of '1' to stop read
i2c_stop();
```

- f. (5 pts) In class, we discussed how the PIC18 *capture* mode could be used to decode both biphasse and space-width encoded IR waveforms. There was a critical difference between how capture mode was used with biphasse decoding and how it was used with space-width encoding, what was this difference?

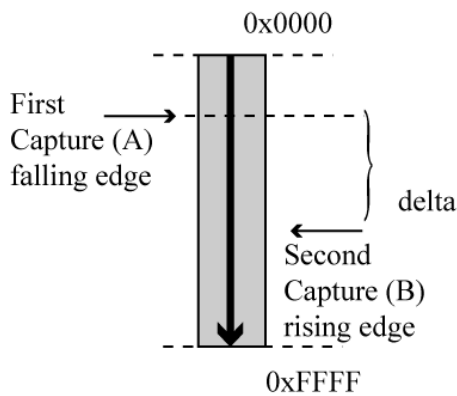
The Biphasse code has to capture time measurements between every edge, the space-width code only captured between falling edges.

- g. (5 pts) For the waveform of problem #c, assume TIMER2 is configured with PRE=16, POST=5, PR2 = 0x90, and FOOSC = 40 MHz. The periodic interrupt that is generated is used to output each point of the waveform. What is the frequency of the waveform in Hz? Show all work.

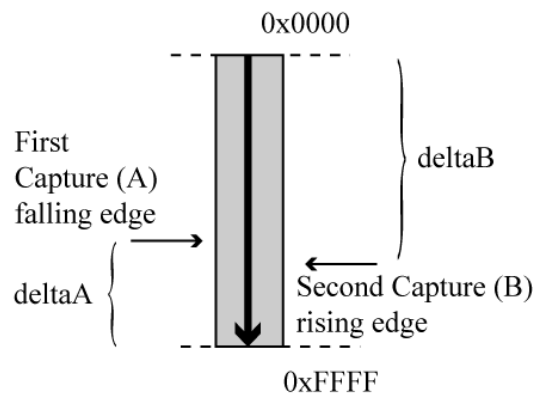
Interrupt period = (PR2+1)* 1/(FOOSC) * 4 * PRE * POST
 = 0x91 * 4/(40 MHz) * 16 * 5 = 1.16 ms.

Period is 1.16 ms * 8 samples for full waveform = 9.28 ms, Frequency is ~ 107.8 Hz

- h. (5 pts) When measuring pulse width using capture mode and timer1, give the equation we used that accounted for timer1 overflow. Draw a diagram that illustrates the components of the equation.



(a) No overflow case
 TimerDelta = B - A



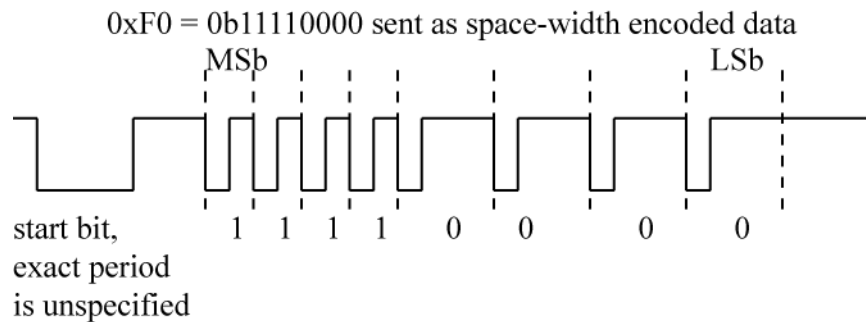
(b) Overflow case
 TimerDelta = (#oflows-1)* 65536 + deltaA + deltaB
 = (#oflows -1) << 2¹⁶ + (0 - A) + B

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. A 3-bit flash A/D has 7 comparators; each whose output is can be either 0 or 1. Assume a $V_{ref} = 4$ V. What is the 7-bit output of these comparators if the input voltage is 2.7 V? Give the 7-bit value in binary, with the LSB the comparator output with the smallest V_{REF} input and the MSB the comparator with the highest V_{REF} input.

The reference voltages on the 7 A/Ds from the resistor string are $7/8 * V_{ref}$, $6/8 * V_{ref}$, $5/8 * V_{ref}$, $4/8 * V_{ref}$, $3/8 * V_{ref}$, $2/8 * V_{ref}$, $1/8 * V_{ref}$.
 The input voltage 2.7 V is between $6/8 * V_{ref}$ (3 V) and $5/8 * V_{ref}$ (2.5 V), so the outputs of the comparators are "0011111".

2. Draw 0xF0 as a space-width encoded waveform, assume '0' (75% duty cycle) is twice the period of a '1' (50% duty cycle); assume MSB is sent first.



3. How many TIMER1 tics equate to a 500 μ s value assuming an FOSC of 35 MHz and a prescale value of 1?

1 Timer1 tic period = $4 / (FOSC) * 1 = 4 / (35\text{MHz})$.
 So, $500 \mu\text{s} / (1 \text{ timer1 tic}) = 500 \mu\text{s} * 35 \text{ MHz} / 4 = 4375$

4. Given a V_{ref} of 4.4 volts, and an input voltage of 2.5 Volts, and a 7 bit A/D, what is the output code (give this in hex) ?

$V_{in} / V_{ref} * 2^7 = 2.5 / 4.4 * 128 = 72.7$, round to 73, so hex code is 0x49

5. What is the minimum output change that you could expect to see on the output of a 6-bit DAC given a V_{ref} of 4.1 V?

$$V_{ref} / 2^6 = 4.1 / 64 = 0.064 \text{ V (64 mV).}$$

6. The Timer1, Timer3 subsystems can be clocked using an external clock, and provide an external oscillator for this clock. We discussed two reasons in class why you might want to use an external clock with Timer1/Timer3, give one of these reasons.

One reason would be to implement a real time clock using an external 32.768 KHz crystal. Another reason would be to use this as a wakeup source from sleep mode since the timer can still function with the internal PIC clocks shut off.

7. How many clock cycles would you expect a 12-bit successive approximation A/D to perform a conversion?

One clock for each bit, so 12 clock cycles.

8. How do you know if an I2C transaction is a read or write transaction?

By the LSB of the first byte of the transaction (the LSB of the address bit). If it is "0", then it is a write operation, else it is a read operation.

9. Write a C code fragment that performs a STOP condition on the I2C bus (do not use the `i2c_stop` function, I want to know what is inside the `i2c_stop` function!).

```
PEN=1;      // initiate stop, PEN=1
//wait until stop finished
while (PEN);
```

10. In some applications using VDD as an A/D reference voltage is fine, but in other cases it is not. When would you not use VDD as an A/D reference voltage?

The power supply is noisy, it also can vary with temperature and current draw. If you do not need much precision, then Vdd is ok as a reference. If you need a high degree of precision in your conversions, then you need a separate voltage reference that is stable with temperature, and current draw, and is noise-free.