

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_getbyte()`, `i2c_stop()`, and `i2c_putbyte()`. If you use `i2c_putbyte`, you must pass in as an argument the byte that is to be written to the I2C bus.

Part I: (64 pts) You must answer all of these questions.

- a. (15 pts) Write C code that will count the number of falling edges of an input waveform and print the result. Assume that the CCP1/TIMER1 capture compare module is being used. Assume the serial port and TIMER1 has already been configured in some manner. Your code has to configure the CCP1 module, and enable both TIMER1 and CCP1 interrupts. Prompt the user to hit any key. After the first falling edge is detected, begin counting falling edges. If no falling edges occur for > 50 timer1 interrupts, then assume the input is idle, and print out a message that contains the number of falling edges that occurred. Use an *int* variable to hold the number of falling edges. Your code must have a clearly identified interrupt service routine; I assume any code outside of the ISR is in *main()*. You must make use of the CCP1 module to detect the falling edges, and your code must be interrupt driven.

```
char print_flag;
int edges;
int tmr1_oflow;

my_isr() {
    if (TMR1IF) {
        TMR1IF = 0; //clear iflag
        tmr1_oflow++;
        if (tmr1_oflow > 50) {
            // input is idle
            print_flag = 1;
            //disable interrupts
            TMR1IE = 0;
            CCP1IE = 0;
        }
    }
    if (CCP1IF) {
        CCP1IF = 0; // clear iflag
        edges++; //found an edge
        tmr1_oflow = 0; //reset
    }
} //end my_isr
```

```
main (){
    // assume TMR1 configured

    CCP1CON = 0x04; // look for falling edges
    while(1) {
        print_flag = 0;
        edges = 0;
        tmr1_oflow = 0;
        printf("Hit any key."); pcrLf();
        getch(); //wait for use to hit key
        // enable interrupts
        IPEN = 0;
        TMR1IE = 1;
        CCP1IE = 1;
        PIE1 = 1;
        GIE = 1;
        while(!print_flag); // wait for idle
        printf("Edge count is %d",edges);
        pcrLf();
    }
}
```

This problem uses concepts covered in the IR detection lab.

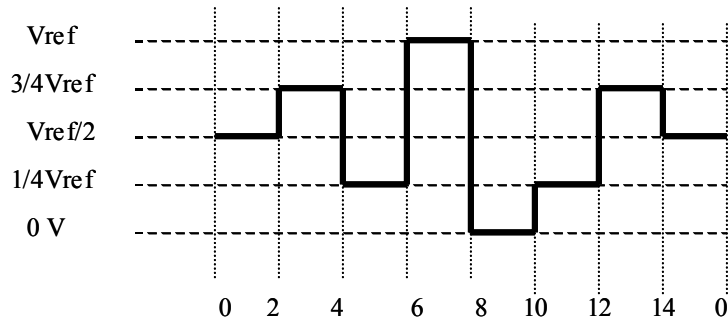
- b. (15 pts) Assume the PIC A/D has been configured to be left-justified, with an external reference voltage of 4.1V. Also assume that the serial port, and I2C bus has been configured. Write a loop that continually performs A/D conversions, and prints out the message “OUT OF RANGE” if the A/D input is lower than 0.5V or higher than 3.4V. Ignore the lower 2 bits of the A/D conversion value.

```
0.5 V/ 4.1 V * 256 = 31.2, round to 31 = 0x 1F  
3.4V/ 4.1 V * 256 = 212.3, round to 212 = 0xD4
```

```
unsigned char c;  
// assume A/D already configured  
while (1) {  
    bitset(ADCON0, 2); // start conversion  
    while (bittst(ADCON0,2)); //wait for conversion to finish  
    c = ADRESH; // left-justified, just get upper 8 bits.  
    if ((c < 0x1F) || (c > 0xD4)) {  
        printf("Out of range"); pcrLf();  
    }  
}
```

This problem uses concepts covered in the A/D lab.

- c. (9 pts) We want to generate the waveform below using a 16-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 16 element char array used to

`char my_table[] = { ??? } // 16 entries.`

To make it easier to grade, fill in the table below with your 16 entries (the array indices are shown). Write your values in hex; this is the value to write to the DAC.

$$\begin{aligned}
 3/4V_{ref} &= 0.75 * 5 \text{ V} = 3.75 \text{ V}. & 3.75 / 5 \text{ V} * 256 &= 192 = 0xC0 \\
 1/2V_{ref} &= 0.5 * 5 \text{ V} = 2.5 \text{ V}. & 2.5 / 5 \text{ V} * 256 &= 128 = 0x80 \\
 1/4V_{ref} &= 0.25 * 5 \text{ V} = 1.25 \text{ V}. & 1.25 / 5 \text{ V} * 256 &= 64 = 0x40
 \end{aligned}$$

0: 0x80	4: 0x40	8: 0x00	12: 0xC0
1: 0x80	5: 0x40	9: 0x00	13: 0xC0
2: 0xC0	6: 0xFF	10: 0x40	14: 0x80
3: 0xC0	7: 0xFF	11: 0x40	15: 0x80

This problem uses concepts covered in the A/D, timer lab that generated waveforms.

- d. (10 pts) Write C code that implements the waveform of problem (c). Your main code can assume that TIMER2 has already been configured to generate an interrupt at the desired interval, and that the I2C bus has been configured. Use the MAX517 D/A, and assume both AD1 and AD0 are tied high. The DAC update should be done in the interrupt service routine; assume AD0, AD1 are both tied high.

```
char index;
char my_table[]= { //contents previous problem};

//isr outputs lookup table value to DAC.
interrupt my_isr() { // triggered by TIMR2 interrupt, assume only interrupt
    TMR2IF = 0;
    // output value to DAC
    i2c_start();
    i2c_putbyte(0x5E); //address, 0101 1110
    i2c_putbyte(0x00); // command byte
    i2c_putbyte(my_table[index]); // output lookup table value;
    i2c_stop();

    index++;
    if (index > 15) index = 0; // wrap index back to start
}

main(){
    // assume timer2 is initialized, and interrupts enabled
    printf("Waveform starting..."); pcrLf();
    while(1) {
        // nothing to do, ISR does all work
    }
}
```

This problem uses concepts covered in the A/D, timer lab that generated waveforms.

- e. (15 pts) Assuming the microchip 24C515 EEPROM, write a C subroutine that will return a non-zero value if the serial EEPROM is erased (contents all 0xFFs). If the EEPROM is not erased, then return a value of '0'. Assume that A1 is tied high, and A0 tied low.

```
// there are several ways to solve this problem. All of them involve sequentially
// reading the EEPROM contents, and determining if any byte is not 0xFF.
// You have to remember that you can only sequentially read 32K, as the command
// byte contains the most significant bit of the 16-bit address

char is_erased() {
    char rval;
    rval = check_block(0xA5); // current address read, low block
    if (!rval) return(rval); // return if low block not erased
    rval = check_block(0xAD); // current address read, high block
    return(rval);
}

//this checks if a 32K block is erased.
char check_block(cmd)
    char cmd;
{
    int i, byte;
    // check all 32K bytes, don't care where we start
    for(i = 0; i < 32768; i++) {
        // do current address read
        i2c_start();
        i2c_putbyte(cmd);
        byte = i2c_getbyte();
        i2c_stop();
        if (byte != 0xFF) return(0) //location not erased
    }
    return(1); //if get here, all bytes erased.
}
```

This problem uses concepts covered in the I2C, serial EEPROM lab.

Part II: (36 pts) Answer 12 out of the next 15 questions. Cross out the 3 questions that you do not want graded. Each question is worth 3 pts, there is no partial credit.

1. How many comparators are required for a 4-bit FLASH A/D?

2^{N-1} comparators needed, $N=4$, so 15 comparators needed.

2. How were ones and zeros encoded in the IR waveform?

Space-width modulation, ones and zeros had full clock waveforms, but different periods.

3. How is the basic storage cell of a DRAM implemented?

One transistor and one capacitor

4. What is the advantage of a DRAM over SRAM?

DRAM is denser than SRAM.

5. We looked at a PIC system that had 4 external SRAMs. What was the key pin on the SRAM that allowed us to build a system with multiple SRAM devices?

Chip Select

6. Give a PR2 and prescale values for a 8 KHz square wave generated using the PWM mode assuming an $F_{osc} = 15$ Mhz. Choose the prescale so that PR2 is greater than 10.

let Prescale = 4, $PR2 = [15e6 / (8e3 * 4 * 4)] - 1 = 116.2 = 116 = 0x74$
let Prescale = 16, $PR2 = [15e6 / (8e3 * 4 * 16)] - 1 = 28.2 = 28 = 0x1C$

7. Assume a timer2 interrupt is being generated using a postscaler of 5, prescale of 4, and a PR2 value of 0x3F (63). How can these values be changed to generate an interrupt that has a period 10 times longer?

Need a ratio where $\text{New } ((\text{PR2}+1) * \text{PRE} * \text{POST}) / \text{Old } ((\text{PR2}+1) * \text{PRE} * \text{POST}) = 10$.
If set new PRE to 16, keep POST the same, we have $((\text{PR2}_{\text{new}}+1) * 16 * 5) / (64 * 4 * 5) = 10$;
so $\text{PR2}_{\text{new}}+1 = (10 * 64 * 4) / 16 = 160$, so $\text{PR2}_{\text{new}} = 159$ New values: POST=5, PRE = 16, PR2 = 159

8. Given a Vref of 4 volts, and an input voltage of 1.3 Volts, and a 6 bit A/D, what is the output code (give this in hex) ?

$$1.3\text{V} / 4\text{V} * 2^6 = 20.8 = 21 = 0x14$$

9. What is the minimum output change that you could expect to see on the output of a 7-bit DAC given a Vref of 5V?

$$1/2^7 * 5\text{V} = 1/128 * 5\text{V} = 0.039\text{V}$$

10. In PWM mode, if CCPRL1 is 0x30, and PR2 is 0xB0, what is the duty cycle of the square wave?

$$0x30 = 48, 0xB0 = 176. \quad 48/176 = 0.27, \text{ so } 27\%$$

11. Why is a sample and hold circuit necessary for a successive approximation A/D?

A successive approximation A/D takes several clocks to convert; the sample/hold circuit keeps the input voltage stable during the conversion.

12. Assuming an FOSC of 10 Mhz, what value has to be written to SSPADD to get a 100 KHz I2C rate?

$$\text{SPADD} = [10\text{e}6 / (100\text{e}3 * 4)] - 1 = 24 = 0\text{x}18$$

13. Write a C code fragment that performs a START condition on the I2C bus.

```
bitset(SSPCON2,0); // begin start
while(bittst(SSPCON2,0)); // wait for end
```

14. Write C code that configures the timer2 system to have a prescale of 4, postscale of 10, and enables the timer2 to be on.

```
                // 1:10 on pre=4
T2CON = 0x4D; // 0 1001 1 01
(you have to be careful for this, a 1:10 prescale has bit code 1001, not 1010).
```

15. Assume the current through an LED being driven by a PWM square wave is 8 mA at duty cycle of 70%. If the duty cycle is changed to 30%, what would I expect the current through the LED to be?

$$\text{new current} = 30/70 * 8 \text{ mA} = 3.4 \text{ mA}$$