CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

ECE 341L Fall 2017 Session 3 Microcontroller Lab ECE341L

Felix Pinai

Experiment 8

LED Light Controls through switches and Serial Port

Build the LED circuit as shown on the attached schematics. This light display will have 8 different light patterns that it can generate. See the description of these patterns at the end of this document. Implement the software that will perform the following tasks:

1) The software will start the display with the first pattern and keep running that pattern until a change is forced through switches or serial port (see below).

2) The software can switch from one mode to another by the use of two 'UP' or 'DOWN' push-button switches. The display will change to a new pattern after the switch is pressed and only after it is released. The display modes go from 0 to 7 and wrap around in both directions.

3) A new display can be also selected through the use of the serial port. A menu is shown on the TeraTerm terminal with a selection of 8 different displays ranging from 0 to 7. Once the user has selected a choice, the software will run that display. The same menu will be displayed again for the user to enter another choice. While no choice is entered, the software will keep running the selected display.

4) Every time a new choice is selected, a beeper will be activated and a number of beeps is generated to indicate what selection is being run. No sound should be generated after the first pass of the light sequence. There are two types of beep tones: a short one and a long one. The **short tone lasts half a second** while the **long one is one second long**. The frequency of the short tone is 3000Hz with a 50% duty cycle while the long tone has a frequency of 1500Hz and 33% duty cycle. The number of beep tones depends on the number of the sequence and the sound does alternate from short to long with the short tone being always the first one generated. For example, if the sequence is number 5, then the series of beeps will be short-long-short-long-short. For the sequence 0, do generate a series of two short beeps.

5) The sequence number is also output to three LEDs to show its binary value (from 0 to 7).

6) Use a basic timer as the main heart beat for the timing sequence. The basic timer interrupt will occur at a programmable interval 'Tint'. Once the interrupt arrives, set a 'flag' to 1 to indicate that the interrupt has occurred. In the main program, monitor this 'flag' until it is set to 1. When the flag is detected as 1, reset it to 0 and get the software to advance in the generation of the timing sequence.

7) As mentioned in part 6), the interval of the timer interrupt, Tint, is programmable. Use a voltage input Vin at the pin AN0 (RA0) to select the period for the interrupt. A photo resistor is used to change the value of Vin. At the same time, a RGB LED D21 (see schematics) is used to indicate the mode that Tint is on. The following table will specify the value of Tint depending on Vin and the associated color for the RGB LED:

    a. 0V < Vin <= 2V, Tint = 50 msec and D21 is set to BLUE
    b. 2V < Vin <= 3V, Tint = 100 msec and D21 is set to GREEN
    c. 3V < Vin, Tin = 200 msec and D21 is set to RED

Here are the descriptions of the sequences:

Sequence 0:
State 0: All LEDs are on
State 1: All LEDs are off

After State 1, go back to State 0

Sequence 1:
State 0: All LEDs are on
State 1: All LEDs are on
State 2: All LEDs are off
State 3: All LEDs are off

After State 3, go back to State 0

Sequence 2:
State 0: LEDs D1, D3, D5, D7, D9, D11, D13 and D15 are on, the other LEDs are off
State 1: All LEDs are off
State 2: LEDs D2, D4, D6, D8, D10, D12, D14 and D16 are on, the other LEDs are off
State 3: All LEDs are off

After State 3, go back to State 0

Sequence 3:
State 0: LEDs D1, D2, D3, D4, D5, D6, D7 and D8 are on, the other LEDs are off
State 1: All LEDs are off
State 2: LEDs D9, D10, D11, D12, D13, D14, D15 and D16 are on, the other LEDs are off
State 3: All LEDs are off

After State 3, go back to State 0

Sequence 4:
State 0: LEDs D1, D2, D11, D12, D13, D14, D15 and D16 are on, the other LEDs are off
State 1: All LEDs are off
State 2: LEDs D3, D4, D5, D6, D7, D8, D9 and D10 are on, the other LEDs are off
State 3: All LEDs are off

After State 3, go back to State 0

Sequence 5:
State 0: LEDs D1, D2, D3, D4, D9, D10, D11 and D12 are on, the other LEDs are off
State 1: All LEDs are off
State 2: LEDs D5, D6, D7, D8, D13, D114, D15 and D16 are on, the other LEDs are off
State 3: All LEDs are off

After State 3, go back to State 0

Sequence 6:
State 0: LED D1 is on, the other LEDs are off
State 1: All LEDs are off
State 2: LED D2 is on, the other LEDs are off
State 3: All LEDs are off
State 4: LED D3 is on, the other LEDs are off
State 5: All LEDs are off
State 6: LED D4 is on, the other LEDs are off
State 7: All LEDs are off
State 8: LED D5 is on, the other LEDs are off
State 9: All LEDs are off
State 10: LED D6 is on, the other LEDs are off
State 11: All LEDs are off
State 12: LED D7 is on, the other LEDs are off
State 13: All LEDs are off
State 14: LED D8 is on, the other LEDs are off
State 15: All LEDs are off
State 16: LED D9 is on, the other LEDs are off
State 17: All LEDs are off
State 18: LED D10 is on, the other LEDs are off
State 19: All LEDs are off

State 20: LED D11 is on, the other LEDs are off
State 21: All LEDs are off
State 22: LED D12 is on, the other LEDs are off
State 23: All LEDs are off
State 24: LED D13 is on, the other LEDs are off
State 25: All LEDs are off
State 26: LED D14 is on, the other LEDs are off
State 27: All LEDs are off
State 28: LED D15 is on, the other LEDs are off
State 29: All LEDs are off
State 30: LED D16 is on, the other LEDs are off
State 31: All LEDs are off

After State 31, go back to State 0

Sequence 7:
State 0: LED D1 is on, the other LEDs are off
State 1: All LEDs are off
State 2: LED D2 is on, the other LEDs are off
State 3: All LEDs are off
State 4: LED D3 is on, the other LEDs are off
State 5: All LEDs are off
State 6: LED D4 is on, the other LEDs are off
State 7: All LEDs are off
State 8: LED D5 is on, the other LEDs are off
State 9: All LEDs are off
State 10: LED D6 is on, the other LEDs are off
State 11: All LEDs are off
State 12: LED D7 is on, the other LEDs are off
State 13: All LEDs are off
State 14: LED D8 is on, the other LEDs are off
State 15: All LEDs are off
State 16: LED D9 is on, the other LEDs are off
State 17: All LEDs are off
State 18: LED D10 is on, the other LEDs are off
State 19: All LEDs are off
State 20: LED D11 is on, the other LEDs are off
State 21: All LEDs are off
State 22: LED D12 is on, the other LEDs are off
State 23: All LEDs are off
State 24: LED D13 is on, the other LEDs are off
State 25: All LEDs are off
State 26: LED D14 is on, the other LEDs are off
State 27: All LEDs are off
State 28: LED D15 is on, the other LEDs are off
State 29: All LEDs are off

State 30: LED D16 is on, the other LEDs are off
State 31: All LEDs are off
State 32: LED D15 is on, the other LEDs are off
State 33: All LEDs are off
State 34: LED D14 is on, the other LEDs are off
State 35: All LEDs are off
State 36: LED D13 is on, the other LEDs are off
State 37: All LEDs are off
State 38: LED D12 is on, the other LEDs are off
State 39: All LEDs are off
State 40: LED D11 is on, the other LEDs are off
State 41: All LEDs are off
State 42: LED D10 is on, the other LEDs are off
State 43: All LEDs are off
State 44: LED D9 is on, the other LEDs are off
State 45: All LEDs are off
State 46: LED D8 is on, the other LEDs are off
State 47: All LEDs are off
State 48: LED D7 is on, the other LEDs are off
State 49: All LEDs are off
State 50: LED D6 is on, the other LEDs are off
State 51: All LEDs are off
State 52: LED D5 is on, the other LEDs are off
State 53: All LEDs are off
State 54: LED D4 is on, the other LEDs are off
State 55: All LEDs are off
State 56: LED D3 is on, the other LEDs are off
State 57: All LEDs are off
State 58: LED D2 is on, the other LEDs are off
State 59: All LEDs are off
State 60: LED D1 is on, the other LEDs are off
State 61: All LEDs are off

After State 61, go back to State 0

Development Steps:

A) LED Testing

To make sure that all your LEDs are properly connected to the assigned pins, write a simple ‘while (1)’ loop that will output a fixed pattern on all the ports that are interfacing to the LEDs, followed by a time delay, then do a 1’s complement of the original pattern, output the new pattern to the port again, and then followed by the same amount of delay.

For example, we can output the pattern 0x5555 to the all 16 LEDs. Based on the schematics, those 16 LEDs are partitioned into three groups:

a) The first four LEDs are in PORT A and they are at bits 2 through 5. We want to output the value ‘5’ to those LEDs to get the binary pattern 0b0101. However, the first LED used starts at bit 2. We will need to shift that pattern by two bits to the left.
b) The next four LEDs are in PORT B and also from bits 2 through 5. We do need to do the same as part a)
c) The next 8 LEDs are totally in PORT D and we can use the value 0x55.

The ‘while (1)’ loop should be something like:

```
while (1)
{
                                // 7 6 5 4 3 2 1 0
    PORTA = 0x05<<2;            // x x 0 1 0 1 x x
    PORTB = 0x05<<2;            // x x 0 1 0 1 x x
    PORTD = 0x55;               // 0 1 0 1 0 1 0 1

    for (i=0;i<10000;i++);      // delay
                                // 7 6 5 4 3 2 1 0
    PORTA = 0x0A<<2;            // x x 1 0 1 0 x x
    PORTB = 0x0A<<2;            // x x 1 0 1 0 x x
    PORTD = 0xAA;               // 1 0 1 0 1 0 1 0

    For (i=0;i<10000;i++);      // delay
}
```

B) Tone Generation

We will use PWM (Pulse-Width-Modulation) to generate the pulse needed to excite the buzzer. If you are not familiar with the setting of the PWM register, use the following link:

http://www.micro-examples.com/public/microex-navig/doc/097-pwm-calculator.html to determine the settings of the required registers.

Based on the provide parameters for the short and long beeps, enter the value of 4 Mhz for the speed of the processor then the required frequency of your PWM followed by the duty cycle. Once the program have provided you with the registers then write the following routines for the short beep and the long beep:

```
void gen_short_beep (void)
{
        PR2        = ???;              // values for short beep
        CCPR1L     = ???;
        CCP1CON = ???'
        T2CON      = ???;              // Turn T2 on here
        wait_half_sec();               // Wait half-second
        T2CON      = ???;             // Turn T2 off here
        wait_half_sec();
}

void gen_long_beep (void)
{
        PR2        = ???;              // values for long beep
        CCPR1L     = ???;
        CCP1CON = ???'
        T2CON      = ???;              // Turn T2 on here
        wait_one_sec();                // Wait half-second
        T2CON      = ???;             // Turn T2 off here
        wait_half_sec();
}
```

Once you have finished generating those two routines, write a generic program that has a 'while(1)' loop calling both the short and the long beep routines. Don't forget to initialize the output pins and make sure that your processor runs at 4 MHz (OSCCON = 0x60)

After the beep routines are properly working, generate then a new routine that will play the series based on the pattern's number:

```
void play_series_beep_tone(char sequence_number)
```

This routine is simply a FOR loop that starts with the index from 0 up to the 'pattern_number' and it will test whether the index is even or odd and play the appropriate tone (short or long). Do handle the special case where the sequence number is 0 because two short beeps are to be generated.

C) Variable Timer

1) Main program:

   a) Initialization:

      Program your TIMER 0 as you have done in lab #7 with the exception that you will need to use the prescaler this time due to the fact that the timing is at least 100 msec. Set some dummy values into TMR0L and TMR0H and program to enable the timer interrupt and start the timer.

   b) Main loop:

      * Call a routine to read the voltage Vin of Photo resistor.
      * Call a routine to check the value of Vin. That routine should return one value that can be 0,1 or 2 for the three different mode of Tint.
      * From the returned value, determine the hex values needed for each timing mode (100 msec, 200 msec and 300 msec).
      * Save the upper byte and the lower byte into two variables called for example T0L and T0H.

2) T0ISR:

   a) As always clear the hardware interrupt flag
   b) Load the content of T0L into TMR0L and T0H into TMR0H.
   c) Flip the value of output for the Timer LED (TPULSE = ~TPULSE). TPULSE is the LED called Pulse on the schematics.
   d) Set a software flag called TMR_flag to 1. This flag will be used later in the generation of the pattern.

D) Serial Port Transmit/Receive Operations:

At the end of this section is an example of receive interrupt handler and here are some explanations:

1) Main program:

   a) Initialization: Clear the interrupt flag for the RX Interrupt and set the RX Interrupt Enable bit to 1.

   b) While (1) loop: Wait for the 'RX_flag' to be set to 1. Once detected as 1, clear 'RX_flag', print the received character and reprint the menu. You need to add the code to check whether the received character is in the range of your allowable choices. If yes, then set the sequence number to the new one. If no, then printout the message saying that the choice is not correct.

2) chkisr() routine: Check if RC flag is set. If yes, jump to Serial_RX_ISR(). You do need to have checks for other types of interrupts here by adding more if statements.

3) Serial_RX_ISR(): When the receive interrupt occurs, control will be transferred here whereas the Receive character is read and stored into the variable 'RX_char' to be used later and the 'RX_flag' is set to 1 to indicate that a receive interrupt has been received.

```
#include <p18f4321.h>

#include <stdio.h>
#include <math.h>
#include <usart.h>

void chk_isr(void);                        // chk_isr interrupt subroutine prototype
void do_init(void);
void init_UART(void);
void Serial_RX_ISR(void);
void do_print_menu(void);

char RX_char;
char RX_flag;
char sequence;

void init_UART()                           // Initialized UART
{
   OpenUSART (USART_TX_INT_OFF & USART_RX_INT_OFF &
         USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
         USART_BRGH_HIGH, 25);
   OSCCON = 0x60;
}

void interrupt high_priority chk_isr(void)
{
         if (PIR1bits.RCIF == 1)
         Serial_RX_ISR();                  // If RCIF is set it goes to Serial_RX_ISR
                                           // Add code here to check other interrupt

}
void Serial_RX_ISR(void)
{
         RX_char = RCREG;                  // read the receive character from Serial Port
         RX_flag = 1;                      // Set software RX_flag to indicate reception
                                           // of rx character
}
```

```
void do_init(void)
{
        init_UART();                        // Initialize UART
        TRISC=0x80;                         // Make sure that PORT C bit 7 is input

        RX_char = RCREG;                    // read RCRG just to clear it
        RX_flag = 0;                        // Initialize software RX_flag to zero

        PIR1bits.RCIF = 0;                  // Clear the RX Receive flag
        PIE1bits.RCIE = 1;                  // Enables Rx Receive interrupt
        INTCONbits.PEIE = 1;                // Enables PE INT for Peripheral Interrupt
        INTCONbits.GIE = 1;                 // Enables Global interrupt

}

void main()
{
        do_init();                          // do all the initialization
        do_print_menu();                    // print the menu

        while (1)                           // Infinite While loop
        {
                if (RX_flag == 1)           // wait until RX_flag is set to 1 indicating rx
                                            // char
                {
                        printf ("%c\r\n\n", RX_char);
                        RX_flag = 0;

                        if ((RX_char >= '0') && (RX_char <= '7'))
                        {
                                sequence = RX_char - '0';
                        }

                        else
                        {
                                printf ("\n\n*** INVALID ENTRY ***\n\n");
                        }

                        do_print_menu();
                }
        }
}
```

```
void do_print_menu(void)
{
        printf ("\r\n\n     Menu\r\n\n");
        printf ("Choose pattern from 0 to 7\r\n");
        printf ("\r\nEnter choice : ");
}
```

E) External Input Interrupts:

On the schematics, there are two push-button switches connected to Port B bit 0 and bit 1. These switches are the inputs for the external interrupts INT0 and INT1. When either signal transitions from low to high or from high to low (programmable), an interrupt will be generated to the processor. Therefore, it would be very convenient to use these INT0 and INT1 signals to monitor the status of the UP and DOWN push-buttons. By simply setting a condition to generate the interrupt when the switch is pushed and then released (therefore a transition from low to high), you should have to monitor a software flag called for example 'SWU_flag' and when it is set, you would know that the interrupt did happen. The next step is to simply increment the sequence number. You will need to take care of the following:

Main program:

a) Initialization:

1) Clear the interrupt flag for the INT0 and INT1 in the INTCON and INTCON3 registers:

```
INTCONbits.INT0IF          // INT0 IF is in INTCON
INTCON3bits.INT1IF         // INT1 IF is in INTCON3
```

2) Set the interrupt enable for the INT0 and INT1 in the INTCON and INTCON3 registers:

```
INTCONbits.INT0IE          // INT0 IE is in INTCON
INTCON3bits.INT1IE         // INT1 IE is in INTCON3
```

3) Program the type of edge interrupt required for INT0 and INT1 in the INTCON and INTCON3 registers:

```
INTCON2bits.INTEDG0        // Edge programming for INT0 and INT1
INTCON2bits.INTEDG1        // are in INTCON2
```

b) In the While (1) loop: Wait for the 'SWU_flag' or ‘SWD_flag’ variable to be set to 1. Once detected as 1, clear 'SWU_flag' and ‘SWD_flag’ and update up or down the sequence number. Remember when up counting the sequence and it reaches the value of 8, that number should be reset to 0. Similarly, when down counting and when the sequence is 0, the next sequence should be then 7.

c) chkisr() routine:

* Check if INT0IF flag is set. If yes, jump to an INT0 Interrupt Service Routine INT0_ISR() to take care of INT0 interrupt.

* Check if INT1IF flag is set. If yes, jump to an INT1 Interrupt Service Routine INT1_ISR() to take care of INT1 interrupt.

* INT0_ISR() routine: Set the variable 'SWU_flag' to 1.

* INT1_ISR() routine: Set the variable 'SWD_flag' to 1.

F) Pattern Generation

To generate the required patterns, you should start to make the easy ones first and then slowly after you have acquired the process you can implement the harder ones.

First, go back to the 'Variable Timer' section C) above. If you have not to implement the variable timer, then start with a simple one first by forcing a fixed timing (hard code the value of TMR0L and TMR0H). Let us use a 300 msec timer first. Determine what value you should use for your TIMER0's three registers T0CON, TMR0L and TMR0H. Go back to the lab#7 and use it as a reference. You will need to figure T0CON and especially to use the prescaler this time because the timing is now in the order of 300 msec. I would like to suggest the use of the prescaler of at least x8. Figure out the values needed for TMR0L and TMR0H to achieve 300 msec.

T0_ISR:

a) As always clear the interrupt flag (INTCONbits.TMR0F)
b) Reload the content TMR0L and TMR0H with the required values
c) Flip the value of output for the Timer LED (TPULSE = ~TPULSE).
d) TMR_flag = 1;

Where TPULSE is a '#define' statement to declare the use of the PORTC pin that has a LED connected to it (see schematics).

If you have successfully implemented this section, your 'TPULSE' LED should blink at the desired rate (use a scope to check the period of your waveform at the LED).

The 'd)' item above set a software flag 'TMR_flag' to 1. In your main loop, besides the monitoring of the 'RX_flag' and the 'SWx_flag' variables, you should also add another monitoring task of this 'TMR_flag'. If this flag is set to 1, then call a routine called for example 'Do_Pattern(void)'. In this routine, you should use a 'switch' statement to jump to the proper routine that handles each pattern that I have asked for. Here is a typical example:

```
void Do_Pattern(void)
{
switch (sequence)
{
        case 0:                                     //sequence 1
        if (count == 0) LED_Output(0,0);
        else LED_Output(0xff,0xff);
        count++;
        if(count==2) count = 0;
        break;

     case 1:                                        // sequence 2
        if ((count == 0) || (count == 1)) LED_Output(0,0);
        else LED_Output(0xff,0xff);
        count++;
        if(count==4) do_get_ADC();
        break;

     case …

}
```

The routine 'LED_Output()' is as follow:

```
void LED_Output (char Value1, char Value2)
{
        PORTA = (Value1 & 0x0f)  << # of bits where the least significant LED starts in
PORTA
        PORTB = (Value1 & 0xf0) >> # of bits where the least significant LED starts in
PORTB
        PORTD = Value2;      // no need to shift because all the LEDs are used in this port
}
```

In the above example, in the first sequence, since there are two steps in this sequence, we call in step #0 the LED_Output() routine to output all 0 to the LEDs in PORTA, PORTB and PORTD to effectively turn them on. Then, the count is updated to step #1 whereas the LED_Output() routine is now called to output 0xff to all the LEDs in PORTA, PORTB and PORTD causing all LEDs to be turned off. After that step, the count is incremented to become 2 but get reset to 0 forcing the sequence to go back to step 0 to repeat.