CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

ECE 3301L Spring 2019-2 Microcontroller Lab Felix Pinai

LAB 8: Voltage Control Fan Speed with Speed measurement

The goal of this lab is to control the speed of a fan using PWM (Pulse Width Modulation). An input voltage will be measured and based on its value the fan will be changed accordingly. Beside the speed control feature, this lab will also measure the speed that the fan is rotating by capturing the number of tach pulses from the fan. All the data will displayed on a TFT panel.

Before the final implementation of this design, we will need to develop some helper functions needed for the operations.

PART A) Fixed time-based measurements using a timer

On the combo labs #6&7, the routine 'delay_ms (int ms)' was used to generate a delay. Here is the capture of that routine:

```
void delay_ms(int ms)
{
   int count;
   count = (0xffff - COUNT_SCALED) - 1;
   count = count * ms;

        T0CON = 0x04;                        // Timer 0, 16-bit mode, pre scaler 1:32
        TMR0L = count & 0x00ff;              // set the lower byte of TMR
        TMR0H = count >> 8;                  // set the upper byte of TMR
        INTCONbits.TMR0IF = 0;               // clear the Timer 0 flag
        T0CONbits.TMR0ON = 1;                // Turn on the Timer 0
        while (INTCONbits.TMR0IF == 0);      // wait for the Timer Flag to be 1 for done
        T0CONbits.TMR0ON = 0;                // turn off the Timer 0
}
```

We need to modify this routine to come up with a more specific routine called 'delay_500ms(void). This routine will use hardcode values instead of variables. Hence is the basic framework:

```
void delay_500ms(void)
{
        T0CON = 0x04;                        // Timer 0, 16-bit mode, pre scaler 1:32
        TMR0L = 0x??;                        // set the lower byte of TMR
        TMR0H =0x??                          // set the upper byte of TMR
        INTCONbits.TMR0IF = 0;               // clear the Timer 0 flag
        T0CONbits.TMR0ON = 1;                // Turn on the Timer 0
        while (INTCONbits.TMR0IF == 0);      // wait for the Timer Flag to be 1 for done
        T0CONbits.TMR0ON = 0;                // turn off the Timer 0
}
```

You will need to calculate the two values to be loaded into the registers TMR0L and TMR0H in order to generate a pulse that is exactly 500 msec long. Here are some basic settings for the operation of the timer:

- Timer used is Timer 0
- Timer is setup in 16-bit timer
- The system should be running at 8Mhz
- The timer's basic clock is derived (1:4) from the system clock
- A prescaler of 1:32 is used to divide the timer's basic clock

Once that routine is created, you will need to check that does generate the proper waveform that should toggle between high and low every 500 msec. Use this test program:

```
void main(void)
{
	Do_init();				// initialize the I/O (make sure  to setup the direction
						// of the I/O especially the signal 'PULSE'

	OSCCON = 0x70;				// set the CPU speed to be at 8Mhz
	PULSE = 0;				// set the PULSE to be 0 first
	while (1)
	{
		PULSE = ~PULSE;			// Flip the logic state of PULSE
		delay_500ms();			// delay 500 msec.
	}
}
```

PART B) Input Voltage Measurement

Use the program used in the Voltmeter (Lab #5) or any routine that read a voltage like the one that read the voltage of the light sensor, write a routine called '**float Read_Volt_In(void)**'. This routine will return a floating point value on the variable voltage applied at the specified ANx pin shown on the schematics. Don't forget to call the function 'Init_ADC()' at the start of the main program to initialize the various ADCONx registers.

PART C) Fan Speed Control through PWM

The PWM pin of the Fan allows the control of the speed for the fan. By applying a signal with a fixed frequency but with different duty cycle, the fan will rotate at various speeds.

The provided fan can take a PWM pulse with a frequency range from 18Khz to 30 Khz. Let assume that we are going to use 25 Khz as the frequency. Next, let us use the following link:

http://www.micro-examples.com/public/microex-navig/doc/097-pwm-calculator.html

Enter the value of 8 Mhz for the PIC's operating frequency. Next, enter 25000 for frequency of the PWM pulse. This is the frequency required by the fan when PWM is used

By varying the value in duty cycle box from 0 to 99, you should see the value of the registers needed to be modified – PR2, T2CON, CCP1CON and CCPR1L – being changed accordingly.

I have compiled a routine that will change those registers based on a specified value of the duty cycle:

```
void do_update_pwm(char duty_cycle)
{
float dc_f;
int dc_I;
   PR2 = 0b00000100 ;                       // set the frequency for 25 Khz
   T2CON = 0b00000111 ;                     //
   dc_f = ( 4.0 * duty_cycle / 20.0) ;     // calculate factor of duty cycle versus a 25 Khz
                                            // signal
   dc_I = (int) dc_f;                       // get the integer part
   if (dc_I > duty_cycle) dc_I++;           // round up function
   CCP1CON = ((dc_I & 0x03) << 4) | 0b00001100;
   CCPR1L = (dc_I) >> 2;
}
```

The next step is to combine with the work on Part C to control the PWM output based on an input voltage. Complete the following routine:

```
void main()
{
        Do_Init();                                  // initialize all the hardware

        while (1)
        {
                float input_voltage = Read_Volt_In (); // read the input voltage
                char dc = ??;                          // calculate the percentage of the ratio
                                                       // of the input voltage versus the reference
                                                       // voltage of 4.096V. 'dc' must be converted into
                                                       // a 'char'
                do_update_pwm(dc);                     // call routine to generate the PWM pulse
        }
}
```

When the above function is completed, run it and verify using a scope that the waveform of the PWM pulse does change when the input voltage changes. Measure on the scope the duty cycle of the PWM signal.

PART D) Fan Speed measurements

The fan provides a signal called Tach Pulse. This is a square wave signal that goes up and down twice when the fan makes a full revolution. In order to measure the speed of the fan, we just need to count the number of these pulses. If N is the number of pulses measured in a fixed period of times called T, then the number of revolutions per second, called RPS, is equal to:

$$RPS = \frac{1}{2} (N / T)$$

with T measured in second. The '1/2' is to take care of the fact that there are two pulses per revolution
If T= 0.5 sec (or 500 msec), then: $RPS = \frac{1}{2} * (N / 0.5) = N$

This means that if we use a period of time of 500 msec to measure the number of pulses, then that number is actually the RPS.

Next, we will need to write up a function that will capture the RPS value of the fan.

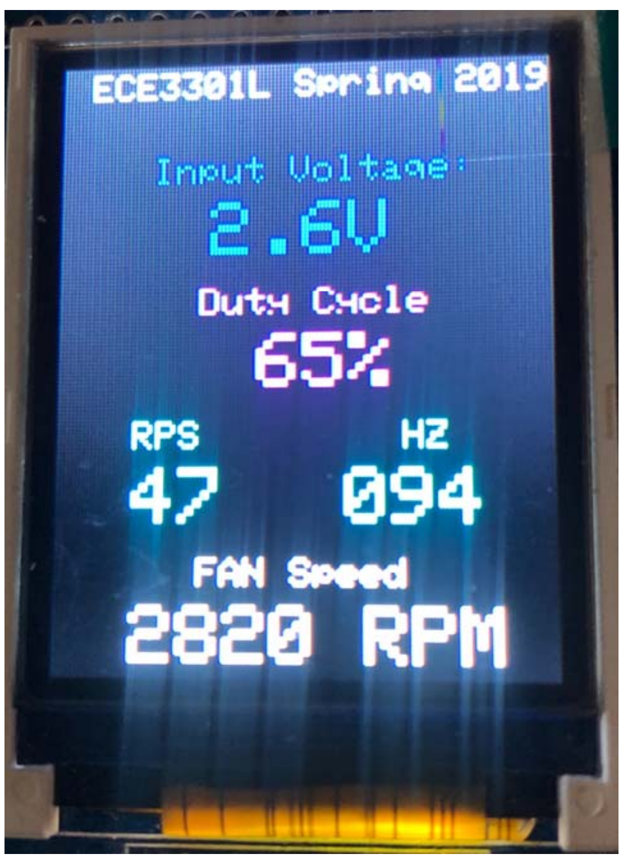
```
int get_RPS(void)
{
   TMR1L = 0;                    // clear TMR1L to clear the pulse counter
   T1CON = 0x03;                 // enable the hardware counter
   PULSE = 1;                    // turn on the PULSE signal
   delay_500ms ();               // delay 500 msec
   PULSE = 0;                    // turn off the PULSE signal
   char RPS = TMR1L;             // read the number of pulse
   T1CON = 0x02;                 // disable the hardware counter
   return (RPS);                 // return the counter
}
```

PART E) Display Information on TFT Panel

We want to display on the TFT screen the following information:

- Voltage Input
- Duty Cycle Applied
- Fan Speed (in RPS, Hz, and RPM)

Here is a screenshot of that display:



The ‘voltage’ group is for the display of the ‘Input Voltage’.
The ‘dc’ group is to show the ‘Duty Cycle’ as the percentage ratio of the input voltage against the reference 4.096V.
The ‘RPS’ group is to display the number of revolutions per second being equal to RPM/60.
The ‘Hz’ group concerns the display of the frequency of the tach pulse in Herz. It should be equal to RPS multiply by 2.
The ‘RPM’ group is for the ‘FAN Speed’.

Below are the data for the locations of the various fields:

```
#define TS_1               1                        // Size of Normal Text
#define TS_2               2                        // Size of Number Text

#define title_txt_X        2                        // X-location of Title Text
#define title_txt_Y        2                        // X-location of Title Text

#define voltage_txt_X      25                       // X-location of Voltage Text
#define voltage_txt_Y      25                       // Y-location of Voltage Text
#define voltage_X          40                       // X-location of Voltage Number
#define voltage_Y          37                       // Y-location of Voltage Number
#define voltage_Color      ST7735_BLUE              // Color of Voltage data

#define dc_txt_X           37
#define dc_txt_Y           60
#define dc_X               52
#define dc_Y               72
#define dc_Color           ST7735_MAGENTA

#define RPS_txt_X          20
#define RPS_txt_Y          95
#define RPS_X              20
#define RPS_Y              107
#define RPS_Color          ST7735_CYAN

#define HZ_txt_X           90
#define HZ_txt_Y           95
#define HZ_X               75
#define HZ_Y               107
#define HZ_Color           ST7735_CYAN

#define RPM_txt_X          37
#define RPM_txt_Y          130
#define RPM_X              20
#define RPM_Y              142
#define RPM_Color          ST7735_WHITE
```

Use the example in the Lab#6&7, use the idea implemented in the routine called ‘Initialize_Screen()’ to modify the screen based on the provided picture and the coordinates above. Below is an extract of the program for this lab. Add the additional codes to complete it.

```
void Initialize_Screen()
{
	LCD_Reset();
	TFT_GreenTab_Initialize();
	fillScreen(ST7735_BLACK);

	strcpy(txt, " ECE3301L Spring 2019\0");
	drawtext(title_txt_X, title_txt_Y, txt, ST7735_WHITE, ST7735_BLACK, TS_1);

	strcpy(txt, "Input Voltage:");
	drawtext(voltage_txt_X, voltage_txt_Y, txt, voltage_Color, ST7735_BLACK, TS_1);

  //-> Place the additional codes to add the information for the following text:
  //->                    * duty cycle
  //->                    * RPS
  //->                    * Hz/
  //->                    * RPM
}

#include "ST7735_TFT.inc"

char *txt;
char buffer[30]       = "";
char voltage_text[]   = "0.0V";
char dc_text[]        = "--%";
char RPS_text[]       = "00";
char HZ_text[]        = "000";
char RPM_text[]       = "0000 RPM";

void main()
{

  init_UART();
  init_ADC();
  init_IO();
  OSCCON = 0x70;

  txt = buffer;

  Initialize_Screen();

  while (1)
  {
    float input_voltage = Read_Volt_In();        // get input voltage
    char dc = ???;                               // calculate duty cycle
    do_update_pwm(dc);                           // generate PWM
    char RPS = get_RPS();                        // measure RPS
    int HZ = RPS * 2;                            // calculate HZ equivalent
    int RPM = RPS * 60;                          // calculate RPM equivalent
```

```
  char iv1 = (int) input_voltage;
  char iv2 = (int) ((input_voltage - iv1) * 10);
  voltage_text[0] = iv1 + '0';
  voltage_text[2] = iv2 + '0';
  drawtext(voltage_X, voltage_Y, voltage_text, voltage_Color, ST7735_BLACK, TS_2);

 // add code for the rest of the fields //

 }


}
```