Fan Speed measurement with Speed control through PWM

The goal of this lab is to control the speed of a fan using PWM (Pulse Width Modulation) by varying an input voltage. 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. The result will be displayed on the TFT panel.

Part A) Fixed time-based measurements using a timer

We will be using the fan that I have provided in the kit along with a wall plug +12V power supply. The fan has four wires:

1) Black wire: Ground pin to be connected to the power supply's ground pin and also to the system's ground.
2) Yellow wire: +12V pin to be connected to the power supply's +12V.
3) Green wire: Tach signal providing a pulse when the fan makes ½ the revolution.
4) Blue wire: PWM wire to control the speed of the fan.

To measure the speed of a fan, we just need to count the number of pulses generated on the Tach signal (green wire). When a fan makes a full revolution, a total of two (2) pulses will be generated. We now use a counter to count the number of pulses within a fixed period of time. Let us use the timer T1 as a counter. To setup it up as a timer, use the datasheet of the PIC18F4620:

http://ww1.microchip.com/downloads/en/devicedoc/39626e.pdf

and go to chapter 12 starting on page 129 and look at the register T1CON. Set the following:

Bit 7: two 8-bit operations – we only need to count up to 256 hence an 8-bit is needed
Bit 6: clock is from another source – the clock comes from the tach pulse fed into the pin T13CKI @RC0 (pin 15)
Bit 5-4: No prescaler used (1:1)
Bit 3: Timer oscillator is shut off
Bit 2: Synchronize to external clock
Bit 1: External clock is from T13CKI
Bit 0: Enable Timer 1

Derive the correct value for T1CON.

We need to write next a routine called int get_RPM() to measure and to return the RPM (revolution per minute) of the fan. We know that RPM = 60 * RPS where RPS is revolution per

second. To get RPS, we can count the number of pulses from the Tach signal per half second and then multiply that number by 2 to get RPS. Let us call the variable tach_pulse as that count, then:

```
int get_RPM()
{
        TMR1L = 0x00;                   // clear the count
        T1CON = 0x??;                   // start Timer1 as counter of number of pulses
        Wait_Half_Second();             // wait half second
        int RPS = TMR1L;                // read the count. Since there are 2 pulses per rev
                                        // and 2 ½ sec per sec, then RPS = count
        return (RPS * 60);              // return RPM = 60 * RPS
}
```

Once completed, do a program to test the count of the fan:

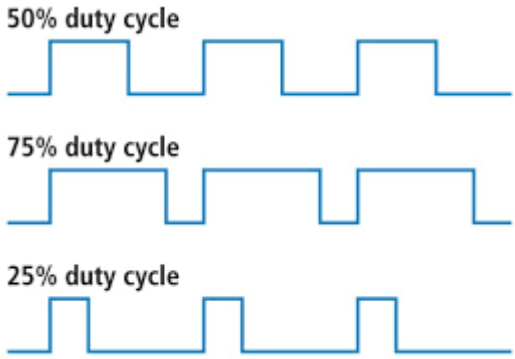
```
void main(void)
{
        Init_IO();
        Init_ADC();
        init_UART();
        OSCCON = 0x70;                  // set the system clock to be 1MHz 1/4 of the 4MHz

        while (1)
        {
                int rpm = get_RPM();
                printf ("RPM = %d\r\n",rpm);
        }
}
```

Make sure when you use TeraTerm to set the baud rate to 19200 and not 9600 since the processor speed is not 8 Mhz and no longer 4 Mhz.

Part B) Fan Speed control

Part B1)

To control the speed of the fan, we are going to use the signal 'PWM_Pulse'. This signal uses the principle of duty cycle to increase or decrease the speed. A signal with a duty cycle has a pulse that is set high for a fixed amount of time with respect to the entire period of the signal. Here are some examples:



The concept of PWM allows the use of duty cycle to vary the speed. A PWM with 50% duty cycle will force the fan to run a half-speed. If PWM is set to 1 (100 % duty cycle), the fan will run at full speed. On the other hand, if PWM is set to 0, then the fan will rotate at its lowest speed.

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;
}
```

Now, we are going to test the use of the PWM. Modify the program above to set different value of duty cycle and check the RPM of the fan based on the set duty cycle:

```
void main(void)
{
        int duty_cycle = 50;
        Init_IO();
        Init_ADC();
        init_UART();
        OSCCON = 0x70;                  // set the system clock to be 1MHz 1/4 of the 4MHz

        do_update_pwm(duty_cycle) ;

        while (1)
        {
                int rpm = get_RPM();
                printf ("RPM = %d\r\n",rpm);
        }
}
```

Change the different value of 'duty_cycle' by modifying the value and rerun the program. Observe that the fan runs faster or slower if 'duty_cycle' is increased or decreased.

Part B2)

The next step is to write a function called 'int get_duty_cycle()' whereas the voltage is read from the pin AN0 (or RA0) and the duty cycle value can be calculated by taking the ratio of that voltage divided by 5V and then multiplied by 100.

```
float read_volt()
{
float volt;
        int nStep = get_full_ADC();
        volt = nStep * 5 /1024.0;
        return (volt);
}

int get_duty_cycle(float volt)
{
        int dc = (int) (read_volt() / 5.0 * 100.0);
        return (dc);
}
```

```
void main(void)
{
        int duty_cycle;
        Init_IO();
        Init_ADC();
        init_UART();
        OSCCON = 0x70;                  // set the system clock to be 1MHz 1/4 of the 4MHz

        while (1)
        {
                volt = read_volt();
                duty_cycle = get_duty_cycle(volt);
                duty_cycle = get_duty_cycle();
                do_update_pwm(duty_cycle) ;
                int rpm = get_RPM();
                printf ("duty cycle = %d RPM = %d\r\n",duty_cycle, rpm);
                Wait_Half_Second();
        }
}
```

Run the program while varying the potentiometer. Observe that when the voltage changes, the duty cycle changes and the RPM also varies accordingly.

Part B3)

When the above task works successfully, then implement two new functions to control the outputs of the two RGB LEDs D1 and D2:

LED D1: void Set_DC_RGB(int duty_cycle)
LED D2: void Set_RPM_RGB(int rpm)

Here are the requirements for the routine Set_DC_RGB(int duty_cycle):

a) If duty cycle = 0, no color to be displayed
b) If duty cycle > 0 and < 33, color is RED
c) If duty cycle >= 33 and < 66, color is YELLOW
d) If duty cycle >= 67, color is GREEN

Here are the requirements for the routine Set_RPM_RGB(int rpm):

a) If rpm = 0, no color to be displayed
b) If rpm > 0 and < 1200, color is GREEN
c) If rpm >=1200 and < 2400, color is CYAN
d) If rpm >= 2400, color is BLUE

Once these routines are implemented, create a routine called 'void Monitor_Fan()' that will combine the following functions:

```
void Monitor_Fan()
{
        volt = read_volt();
        duty_cycle = get_duty_cycle(volt);
        do_update_pwm(duty_cycle) ;
        int rpm = get_RPM();
        Set_DC_RGB(int duty_cycle):
        Set_RPM_RGB(int rpm):
        printf ("duty cycle = %d RPM = %d\r\n",duty_cycle, rpm);
        Wait_Half_Second();
}
```

The while loop in the main program should only call this Monitor_Fan() function to constantly check the voltage, calculate the duty cycle, adjust the PWM, measure the fan speed and display the colors of the two RGB LEDs.

Part C) Fan Operational Control using push-button

Use the two push-buttons and the implementation of the INT0 and INT1 interrupt to turn off (INT0) and to turn on (INT1) the fan.

As implemented in the example provided in lab #9, when either the push-button at INT0 or INT1 is pressed, the INT0_ISR() or the INT1_ISR() is called upon and the variable INT0_flag or INT1_flag is set to 1. Back in the main program and inside the while (1) loop, these two variables are constantly monitor and when one is detected to be '1', then it is subsequently cleared but a message will be printed to show that the associated push-button has been pressed. Now, we want to use both buttons to implement the feature to turn off and to turn on the fan. Use INT0 for the off function and INT1 for the on function. First, we will need to write two new functions:

1) void Turn_Off_Fan():
    a. Set duty cycle to 0
    b. Call function 'do_update pwm()' based on new duty cycle
    c. Call 'Set_DC_RGB(iduty_cycle)';
    d. Set variable rpm = 0
    e. Call 'Set_RPM_RGB(rpm)'
    f. Set variable FANEN = 0
    g. Set FANEN_LED = 0

2) void Turn_On_Fan():
    a. Set variable FANEN = 1
    b. Set FANEN_LED = 1

Once these functions are implemented, take the code from part B3) and add the code to monitor the push-button switches from lab #9 to come up with a new code that will turn off the fan when INT0 is pressed and turn on that fan when INT1 is pressed. The while (1) loop will monitor three variables: INT0_flag, INT1_flag and FANEN. Use the function Turn_Off_Fan() when INT0_flag is detected to be 1. The same applied to INT1_flag and Turn_ON_Fan(). For the variable FANEN, when it is detected to 1, then call the function Monitor_Fan() developed in part B3). When it is 0, simply call the function Wait_Half_Second() back-to-back (to make one full second wait) to do nothing but blinking the SEC_LED.

Part D) Fan Operation Status on LCD

The next step is to provide the information on the LCD display. Attached is the basic framework for the display through the file 'Lab10_sample.c'

We will need to create three functions that will update show the value of the input voltage, the bar graph for the duty cycle, and the graph for the RPM.

D1) On the sample code, the function Update_Volt() is partially provided to read the input voltage and to display that value on the LCD. Complete that function by adding additional codes to place the remaining digits onto the screen.

D2) Produce the function 'draw_bar_graph_dc()' by performing the following steps:

1) Take the value of variable 'duty_cycle' and move it to the variable 'DC_Txt ' so that it can be displayed on the LCD. There are two digits in 'duty_cycle'.
2) A rectangle is drawn and it is filled with the black color (code is already provided)
3) A test to check the value of the variable 'duty_cycle'
    a. If it is 0, do nothing because the rectangle will show blank
    b. If it is less than 33, then fill a RED rectangle with the length of the variable 'duty_cycle'
    c. If it is between 33 and less than 67, then fill a RED rectangle with the length of 33 and next to it another rectangle with the length being the difference of the variable 'duty_cycle' and 33. That rectangle should be in YELLOW
    d. If it is greater or equal to 67, then fill the two rectangles with the length of 33, first one in RED then the next one in YELLOW. Then add a third rectangle next to the GREEN with the length being the difference between 'duty_cycle' and 67.
4) The text of showing the value of 'duty cycle' in 1) must be drawn with the color of the range of that variable. The same applies to the box rectangle that covers the graph.

D3) Perform the same steps as described in the ‘draw_bar_graph_dc()’ to implement the function ‘draw_bar_graph_rpm()’:

1) The value is indicated by the variable ‘rpm’
2) Fill in the text value for the variable RPM_Txt. Four digits must be filled.
3) The variable ‘rpm’ has the range from 0 to 3600. That range must be rescaled in order to bring the scale to be from 0 to 100.
4) Use the same proportions as in the graph for the duty cycle.
5) Replace the RED color by GREEN, YELLOW by CYAN and GREEN by BLUE.
6) The color of the text and the color of the box rectangle takes the color of the highest value.

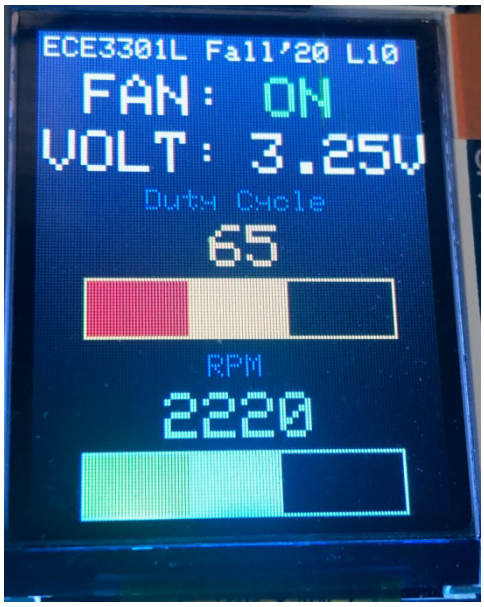
Once these two routines are created, take the function ‘draw_bar_graph_dc()’ and insert it at the beginning of the function ‘Set_DC_RGB(int duty_cycle)’. Do the same for ‘draw_bar_graph_rpm()’ into ‘Set_RPM_RGB(int rpm)’.

Below are some pictures of the implementations using the LCD:

1) FAN OFF

2) FAN ON / LOW SPEED

3) FAN ON / MEDIUM SPEED



4) FAN ON / HIGH SPEED