CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

ECE 3301L **Microcontroller Lab** **Felix Pinai**
Spring 2022 - Session 3

LAB 12: 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 137 and look at the register T3CON. Set the following:

Bit 7: RD16 – We don't need 16-bit operations
Bits 6,3: Ignore these two bits
Bit 5-4: No prescaler used (1:1)
Bit 2: Synchronize to external clock
Bit 1: External clock is from T13CKI
Bit 0: Disable Timer 3 first

Derive the correct value for T3CON. Initialize T3CON in the main initialization routine.

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 second and

since there are 2 tach pulses per revolution, then RPS will be half the amount of pulses counted in one second.

```
int get_RPM()
{
        int RPS = TMR3L / 2;            // read the count. Since there are 2 pulses per rev
                                        // then RPS = count /2
        TMR3L = 0;                      // clear out the count
        return (RPS * 60);              // return RPM = 60 * RPS
}
```

Put this routine in the provided 'Fan_Support.c'.

Next, take the program used on lab 11 and add the line: rpm = get_RPM(); when a new second has been detected. Add to the printf statement the value of RPM to display the speed of the fan.

Note: We need to turn on the fan for this part of the lab. Set the lines:

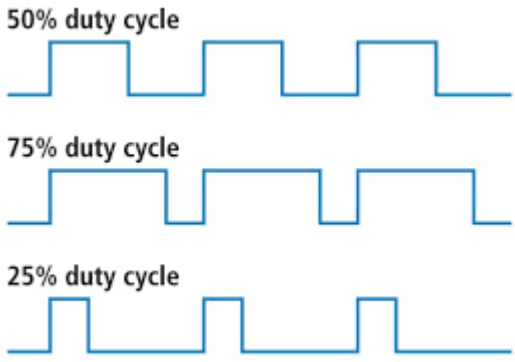
```
FAN_EN = 1;
FAN_PWM = 1;
```

to turn on the FET in order to get the fan going. Look at the schematics to determine what pins are assigned to FAN_EN and FAN_PWM and declare them in 'Main.h' file. We will remove these lines later on.

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;
}
```

Place this routine in the 'Fan_Support.c' file.
Next, go back to the program developed on part A) above. Before, the while(1) loop, place the following lines:

```
        duty_cycle = 50;
        do_update_pwm(duty_cycle) ;
```

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

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 and < 9, color is none
b) If duty cycle >=10 and < 19, color is RED
c) If duty cycle >=20 and < 29, color is GREEN
d) If duty cycle >= 30 and < 39, color is YELLOW
e) If duty cycle >= 40 and < 49, color is BLUE
f) If duty cycle >= 50 and < 59, color is PURPLE
g) If duty cycle >= 60 and < 69, color is CYAN
h) If duty cycle >= 70, color is WHITE

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 < 500, color is RED
c) If rpm >= 500 and < 1000, color is YELLOW
d) If rpm >= 1000 and < 1500, color is GREEN
e) If rpm >= 1500 and < 2000, color is BLUE
f) If rpm >= 2000 and < 2500, color is PURPLE
g) If rpm >= 2500 and < 3000, color is CYAN
h) If rpm >= 3500, color is WHITE

Write those routines without using a multiple IF or case statements. The implementation of each routine should only take no more than 5 lines.

Part C) Fan Operational Control using remote control

Now we are going to use the remote control that we have developed on lab 11 to control the operations of the fan. Three buttons on the remote will be used:

- Button '-' or button number 6
- Button '+' or button number 7
- Button 'Play/Pause' or button number 5

The code for the 'Interrupt.c' done in lab 11 should be moved into this project.

Based on the code on lab 11, use the variable ‘found’ to do the following:

- If found is ‘Play/Pause’ call the function Toggle_Fan()
- If found is ‘-‘ call the function Decrease_Speed()
- If found is ‘+’ call the function Increase_Speed()

Next, we will need to implement those three functions to be placed in the Fan_Support.c file.

- Toggle_Fan():
  - If the variable ‘Fan’ is 1, call function Turn_On_Fan();
  - Else call function Turn_Off_Fan()

  Write up the code for both Turn_On_Fan() and Turn_Off_Fan() to be also placed in the Fan_Support.c file

  - Turn_On_Fan():
    - Set the variable Fan to be 1
    - Call function do_update_pwm(duty_cycle) to set the proper speed
    - Turn on the fan by setting the signal FAN_EN to 1.
    - Turn on the fan LED ‘FAN_LED’

  - Turn_Off_Fan():
    - Set the variable Fan to be 0
    - Turn off the fan using FAN_EN
    - Turn off the fan LED ‘FAN_LED’

- Increase_Speed():

  - Check if duty_cycle is at 100
    - If 100, then generate two beep codes using Do_Beep() function and then reprogram the pwm duty cycle. The duty cycle should remain at 100.
    - If not, then increase duty_cycle by 5 and reprogram the pwm duty cycle

- Decrease_Speed():

  - Check if duty_cycle is at 0
    - If 0, then generate two beep codes using Do_Beep() function and reprogram the duty cycle. The duty cycle must remain at 0.
    - If not, then decrease duty_cycle by 5 and change the pwm duty cycle

Note: The function Do_Beep() is simply the sequence of Activate_Beep, Wait_One_Sec(), Deactivate_Beep(). When done, a call to re-initialize the pwm need to be performed because the Activate_Beep() function will destroy the original pwm.

Run the program and use the remote control to turn on/off the fan and to increase/decrease the speed. Make sure to have the program starts with the fan in the off state (FAN_EN = 0) and the duty_cycle set at 50.

In addition, the RGB D3 should display the color of the button being pressed on the remote control.

Part D) Fan Operation Status on LCD

Make sure that the file 'Main_Screen.c' is now included in the project. Do the following:

1) Place the line Initialize_Screen(); after the line Do_Init();
2) Place the line Update_Screen(); after the printf lines in the while (1) loop
3) Fill the missing lines in the Update_Screen() located in Main_Screen.c file.

Important note: The schematics for this lab have the control lines for the LCD moved to other pins. Make sure to change the pin definitions of those control lines accordingly.

When these tasks are completed, the LCD will display the same information shown on TeraTerm.

Part E) Time Reloading

Add the same code in lab 11 to check when the button ‘EQ’ on the remote is pressed to reload an initial time for the RTC.