Assignment lab 3

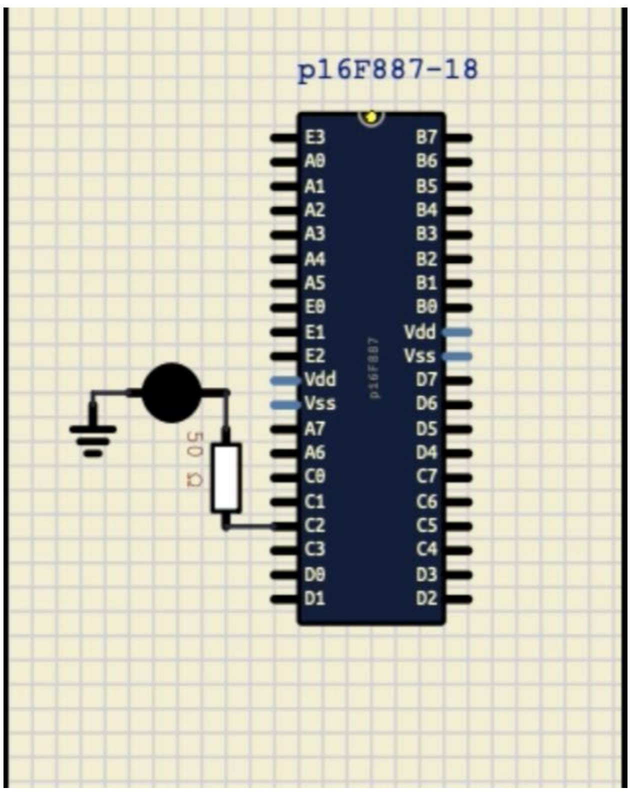
Rashad Mohamed - 202200261

Objective:

In this lab we'll use a simple integration between Timer0 and PWM to control the intensity of a Led and Speed of DC Motor.

PART1

Circuit scheme:



CODE:

```
/*
 * File:   main.c
```

```
 * Author: a
 *
 * Created on March 3, 2025, 9:39 PM
 */



#include <xc.h>
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT enabled)
#pragma config MCLRE = OFF      // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = ON       // Brown-out Reset Selection bits (BOR enabled)
#pragma config IESO = OFF       // Internal External Switchover bit (Internal/External Switchover mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor is enabled)
#pragma config LVP = ON         // Low-Voltage Programming Enable bit (RB3/PGM pin has digital I/O, HV on MCLR must be used for programming)


#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)
void my_delay_ms(unsigned int m_s);
```

```
void __interrupt() ISR(void);

#define prescalar 64.0

#define selected_clock_MHZ 4.0  // 4MHZ


unsigned short pwm_val;

unsigned long overflow_counts = 0;

unsigned long calculated_overflow_counts = 0;


int main() {

   //---[1] configure all pins to be digital [REG : ANSELH and ANSEL]

   OPTION_REG = 0x84; // prescaler is assigned to timer TMR0

   OPTION_REG |= (1 << 2) | (1 << 0);

   ANSEL = 0; // All I/O pins are configured as digital

   ANSELH = 0;

   CCP1CON = 0x0F;  // Select the PWM mode.

   TRISC = 0x00;  // Configure PORTC as output (RC2-PWM1, RC1-PWM2)

   PR2 = 124;

   T2CON |= (1 << 0); // set the prescalar to be 1:4 in the T2CKPS1 and T2CKPS0 pins

   DC1B0 = 0;  // (step6) - set the PWM Duty cycle

   DC1B1 = 0;

   CCPR1L = 0;  // initialize the duty cycle

   TMR2ON = 1;  // Start the Timer for PWM generation

   INTCON = 0xA0;  // Enable interrupt TMR0
```

```
    while (1) {
        my_delay_ms(5000);
        if (pwm_val < 500)
            pwm_val += 100;
        else if (pwm_val == 500)
            pwm_val = 0;
        DC1B0 = (pwm_val & (1 << 0)) >> 0;
        DC1B1 = (pwm_val & (1 << 1)) >> 1;
        CCPR1L = pwm_val >> 2;
    }
}
void __interrupt() ISR(void) {
    overflow_counts++;
    // overflow counts incremented by 1
    TMR0 = 0;
    // Timer TMR0 is returned to its initial value
    INTCON = 0x20;
    // Bit TOIE is set, bit TOIF is cleared
}
void my_delay_ms(unsigned int m_s)
{
    double clk_period = (1 / (selected_clock_MHZ * 1000000.0)); // convert CLK_period to seconds
    double user_period = m_s / (1000.0); // convert user_input to seconds
    unsigned long no_of_counts = (user_period / (4.0 * clk_period * prescalar)); // calculate no. of counts
```

```
needed
   calculated_overflow_counts = no_of_counts / 256; // calculate the overflow counts needed
   TMR0 = 0; // start counting from 0
   while(overflow_counts != calculated_overflow_counts); // wait until overflow counts = the needed
overflow counts
   overflow_counts = 0; // reset
   calculated_overflow_counts = 0; // reset
}
```

PART2

Circuit scheme:

Well we didn’t record anything due the corrupted motor and I could simulate that on simulide.

CODE:

```
#include <xc.h>
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT enabled)
#pragma config MCLRE = OFF      // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = ON       // Brown-out Reset Selection bits (BOR enabled)
#pragma config IESO = OFF       // Internal External Switchover bit (Internal/External Switchover mode is enabled)
```

```
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor is enabled)

#pragma config LVP = ON        // Low-Voltage Programming Enable bit (RB3/PGM pin has digital I/O, HV on MCLR must be used for programming)


#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)

#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)

#define _XTAL_FREQ 4000000


#define MOTOR_IN1 RA3  // Connected to 1A (L293D)

#define MOTOR_IN2 RA4  // Connected to 2A (L293D)

#define ENABLE_PIN RC2 // PWM Output for speed control


void my_delay_ms(unsigned int m_s);

void __interrupt() ISR(void);

#define prescalar 64.0

#define selected_clock_MHZ 4.0  // 4MHZ



void main() {

   ANSEL = 0;

   ANSELH = 0;


   TRISA3 = 0;
```

```
TRISA4 = 0;
TRISC2 = 0;  // PWM

// PWM Config
CCP1CON = 0x0F;
PR2 = 124;
T2CON |= (1 << 0);
CCPR1L = 0;
TMR2ON = 1;

while (1) {
   // Motor Forward
   MOTOR_IN1 = 1;
   MOTOR_IN2 = 0;
   CCPR1L = 50; // 50%
   my_delay_ms(5000);

   // Motor Reverse
   MOTOR_IN1 = 0;
   MOTOR_IN2 = 1;
   CCPR1L = 100; // Full
   my_delay_ms(5000);

   // Stop Motor
```

```
        MOTOR_IN1 = 0;

        MOTOR_IN2 = 0;

        CCPR1L = 0;

        my_delay_ms(3000);

    }

}


void __interrupt() ISR(void) {

    overflow_counts++;

    // overflow counts incremented by 1

    TMR0 = 0;

    // Timer TMR0 is returned to its initial value

    INTCON = 0x20;

    // Bit TOIE is set, bit TOIF is cleared

}

void my_delay_ms(unsigned int m_s)

{

    double clk_period = (1 / (selected_clock_MHZ * 1000000.0)); // convert CLK_period to seconds

    double user_period = m_s / (1000.0); // convert user_input to seconds

    unsigned long no_of_counts = (user_period / (4.0 * clk_period * prescalar)); // calculate no. of counts
needed

    calculated_overflow_counts = no_of_counts / 256; // calculate the overflow counts needed

    TMR0 = 0; // start counting from 0

    while(overflow_counts != calculated_overflow_counts); // wait until overflow counts = the needed
```

```
overflow counts
   overflow_counts = 0; // reset
   calculated_overflow_counts = 0; // reset
}
```