EMBEDDED SYSTEMS FINAL PROJECT REPORT

ECE 47100 Spring 2016

Dr. Stanley Chien

Department of Electrical and Computer Engineering

Indiana University-Purdue University Indianapolis

schien@iupui.edu, (317) 274-2760

Kelly Wachtel

April 26, 2016

TABLE OF CONTENTS

[LIST OF TABLES iv](#_Toc449390833)

[LIST OF FIGURES v](#_Toc449390834)

[LIST OF EQUATIONS vii](#_Toc449390835)

[ABSTRACT viii](#_Toc449390836)

[CHAPTER 1. INTRODUCTION 1](#_Toc449390837)

[CHAPTER 2. PROJECT REQUIREMENTS 2](#_Toc449390838)

[CHAPTER 3. DESIGN DETAILS 3](#_Toc449390839)

[3.1 Hardware 3](#_Toc449390840)

[3.1.1 Motor Types 3](#_Toc449390841)

[3.1.2 Motor Schematic 5](#_Toc449390842)

[3.1.3 Light Emitting Diodes 6](#_Toc449390843)

[3.1.4 Tactile Push Buttons 6](#_Toc449390844)

[3.2 Software 6](#_Toc449390845)

[3.2.1 Data Flow Diagram 7](#_Toc449390846)

[3.2.2 Software Block Diagram 7](#_Toc449390847)

[3.2.3 Flow Diagram 8](#_Toc449390848)

[CHAPTER 4. IMPLEMENTATION DETAILS 9](#_Toc449390849)

[4.1 Parts and Tools 9](#_Toc449390850)

[CHAPTER 5: EXPERIMENT RESULTS 12](#_Toc449390851)

[CHAPTER 6: DISCUSSION 16](#_Toc449390852)

[CHAPTER 7: CONCLUSION 17](#_Toc449390853)

[REFERENCES 18](#_Toc449390854)

[Appendices 19](#_Toc449390855)

[Appendix A: Data Sheets 19](#_Toc449390856)

[D44TD3 Transistor 19](#_Toc449390857)

[Servo Motor 22](#_Toc449390858)

[Appendix B: Code 22](#_Toc449390859)

[Main.c 22](#_Toc449390860)

[Interrupt\_manager.c 23](#_Toc449390861)

[mcc.c 28](#_Toc449390862)

[pin\_manager.c 34](#_Toc449390863)

[pwm4.c 35](#_Toc449390864)

[pwm5.c 37](#_Toc449390865)

[tmr2.c 39](#_Toc449390866)

[Interrupt\_manager.h 46](#_Toc449390867)

[mcc.h 50](#_Toc449390868)

[Pin\_manager.h 53](#_Toc449390869)

[pwm4.h 67](#_Toc449390870)

[pwm5.h 71](#_Toc449390871)

[tmr2.h 75](#_Toc449390872)

[Table 1 Parts and Tools Price List 10](#_Toc449390873)

[Table 2 Software Tools 11](#_Toc449390874)

[Table 3 Standard Servo PWM Wave Chart 13](#_Toc449390875)

LIST OF FIGURES

[Figure 1 Design Schematic 3](#_Toc449390876)

[Figure 2 Servo Motor Schematic 5](#_Toc449390877)

[Figure 3 Light Emitting Diode Schematic 6](#_Toc449390878)

[Figure 4 Tactile Push Button Schematic 6](#_Toc449390879)

[Figure 5 Data Flow Diagram 7](#_Toc449390880)

[Figure 6 Software Block Digram 8](#_Toc449390881)

[Figure 7 Flow Diagram 9](#_Toc449390882)

[Figure 8 Breadboard Layout of Final Project 11](#_Toc449390883)

[Figure 9 97% Duty Cycle 14](#_Toc449390884)

[Figure 10 89.5% Duty Cycle 14](#_Toc449390885)

LIST OF EQUATIONS

[Equation 1 12](#_Toc449390886)

[Equation 2 13](#_Toc449390887)

[Equation 3 13](#_Toc449390888)

ABSTRACT

The purpose of this final project is to simulate windshield wipers using a PIC microcontroller. The final design uses two peripherals available on the microcontroller, Pulse Width Modulation and a timer. The components involved are two servo motors, six tactile push buttons, and six Light Emitting Diodes (LEDs). Windshield wipers operate in two different ways, depending on the car type, in unison or opposing. This paper outlines the design options and implementation.

CHAPTER 1. INTRODUCTION

Windshield wipers are either at zero degrees or 180 degrees. They do not stop in the middle. Depending on the car type, the two wipers may move in unison, both at zero degrees when at location A, and then both at 180 degrees at location B, or in opposition, one at zero degrees and one at 180 degrees when at location A, and then the opposite for location B. This simulation program takes into consideration both of these functionalities. When the program is running, the user can decide which mode to run the “wipers” in, in unison or opposing. From there, the motors speed up or slow down depending on user input. The LEDs light up depending on the speed and mode.

CHAPTER 2. PROJECT REQUIREMENTS

1. Technical
   1. The device shall standalone without corded power.
   2. The device shall use a PIC microcontroller.
   3. The device shall use two peripherals, as discussed in class.
   4. The device shall have a purpose.
2. Purpose
   1. The device shall simulate windshield wipers.
   2. The device shall have two motors.
   3. The device shall have LEDs to show changes.
   4. The device shall be operated by button presses.

CHAPTER 3. DESIGN DETAILS

3.1 Hardware

There are four main hardware components involved in this project. A full description of these components follows. The schematic of the project can be seen in Figure 1.

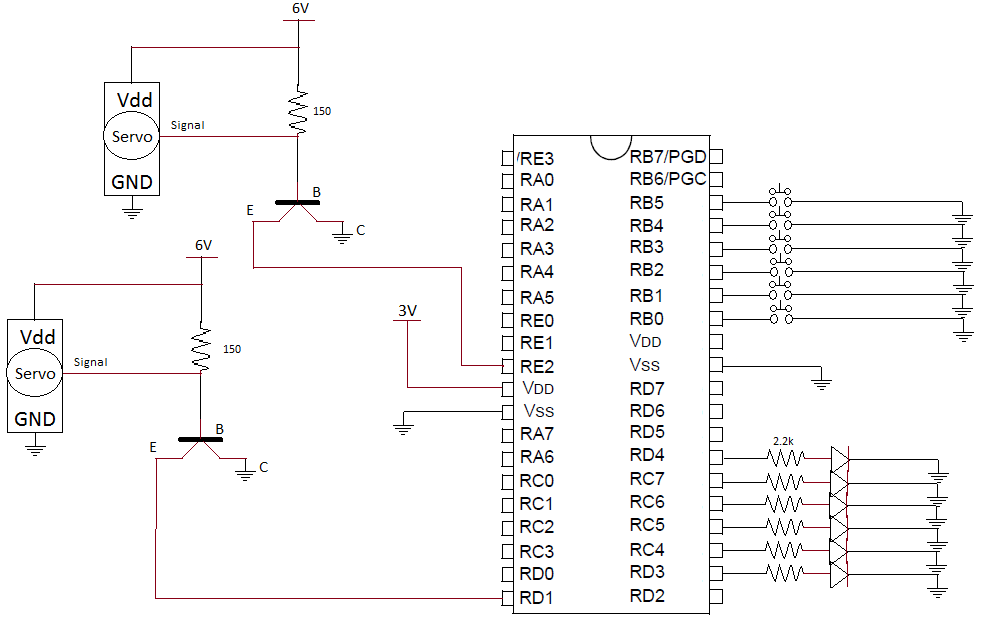


Figure 1 Design Schematic

3.1.1 Motor Types

The design option that needed to be considered for this project was the type of motor that was to be used for the most accurate simulation.

**Stepper Motor**

Advantages:

* Easily controllable.
* Quick setup, only four wires.
* Precise movements
* 360° range of motion

Disadvantages:

* Requires 1.7 Amp current for movement

**Servo Motor**

Advantages:

* 6 V required for movement
* Quick setup, only three wires
* Uses Pulse Width Modulation

Disadvantages:

* Will require a transistor circuit
* 180° range of motion
* Movements not exact

3.1.1.1 Motor Decision

The servo motor proved to be the best decision for this project. Because windshield wipers only need a range of motion from 0° to 180°, there was no need for the stepper motor’s range of motion. After trying to implement the stepper motor, the 1.7 Amp requirement proved to be too high to be easily implemented with just the PIC. The servo motor operates by receiving voltage pulses. This makes it easy to implement the Pulse Width Modulation (PWM) peripheral. The servo motor will require some extra circuitry, but in the end it will do what is needed.

3.1.2 Motor Schematic

The PIC microcontroller chosen for this project is a low power PIC that operates off of 3V. The servo motors, however, require 6V input. This means a transistor circuit is required to allow for two voltage source to be used. The complete schematic can be seen in Figure 2. A 150 Ohm resistor was decided upon to work with the transistor. The NPN transistor used in this operation has a 1.5 V drop across the saturation field. If this is subtracted from the supplied 6 V, the result is 4.5V. From here, the Base-Emitter Saturation current cannot exceed 2 Amps, as described in the data sheet. This meant that the 150 Ohm resistor chosen would suffice. This data sheet is supplied in Appendix A.

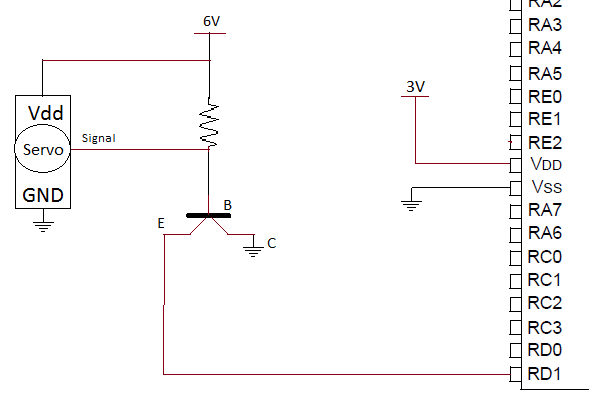


Figure 2 Servo Motor Schematic

3.1.3 Light Emitting Diodes

As the motor speeds up and slows down, LEDs light up to represent a certain speed. A resistor between the pin-out of the chip and the positive end of the diode is vital so the light does not blow out. 2.2 kOhm resistors are used for this purpose. A schematic of one LED connection is seen in Figure 3.



Figure 3 Light Emitting Diode Schematic

3.1.4 Tactile Push Buttons

The buttons used for the selection of modes and speeds were tactile push buttons. The six buttons used were all connected to register 6 pins so that pull up resistors would not be necessary, since the register 6 pins provide internal Weak Pull-Up capabilities. A single connection schematic of a tactile push button can be seen in Figure 4.



Figure 4 Tactile Push Button Schematic

3.2 Software

The software code was written in the C programming language. The MPLAB X IDE was used for the code implementation. Within this IDE, there is a downloadable plug in tool called MPLAB Code Configurator. This tool allows the programmer to visually see the registers set on a PIC microcontroller and also more easily set the peripheral settings. In the case of this project, the two peripherals, PWM and timer, were set up using the Code Configurator. This allows for a more exact calculation of the PWM duty cycle and timing aspects.

3.2.1 Data Flow Diagram

Displayed in Figure 5 is the basic functionality of the program. The diagram begins by the user selecting either Mode 1 or Mode 2.

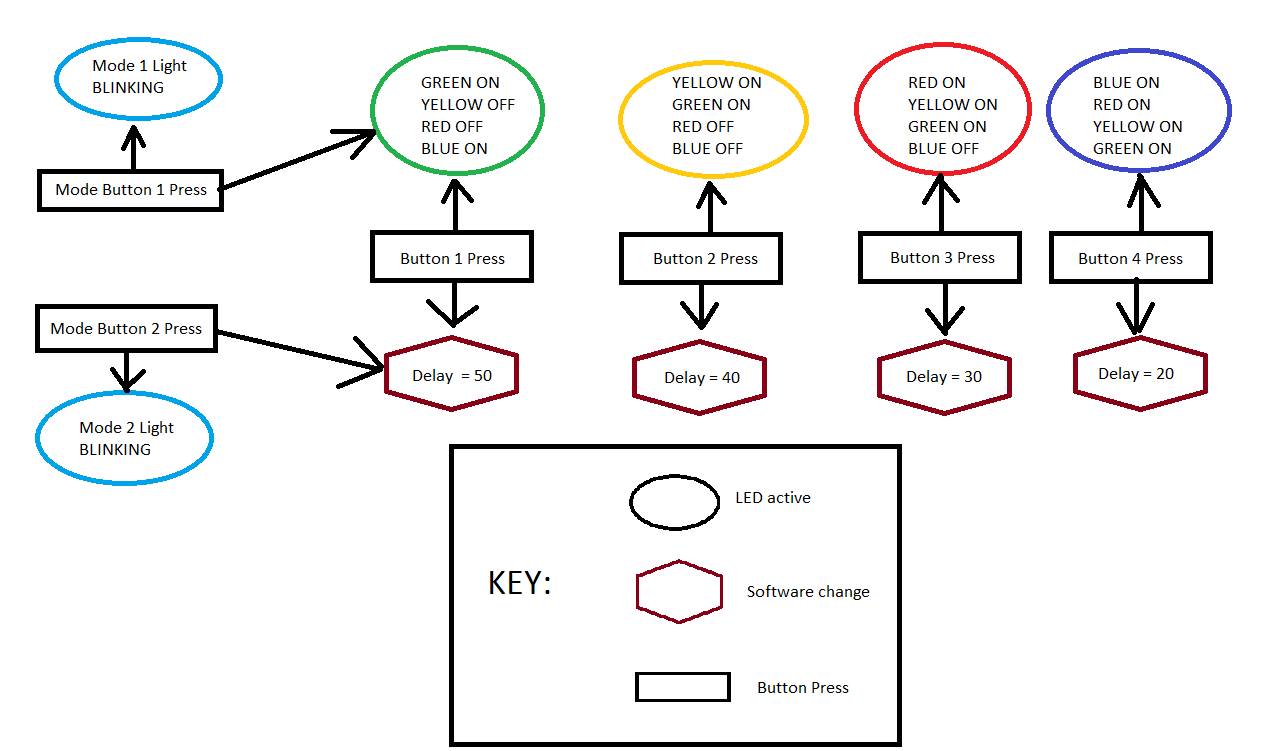


Figure 5 Data Flow Diagram

3.2.2 Software Block Diagram

The software is compiled of five c files that contain all the function definitions required for program setup and execution. This design is outlined below in Figure 6.

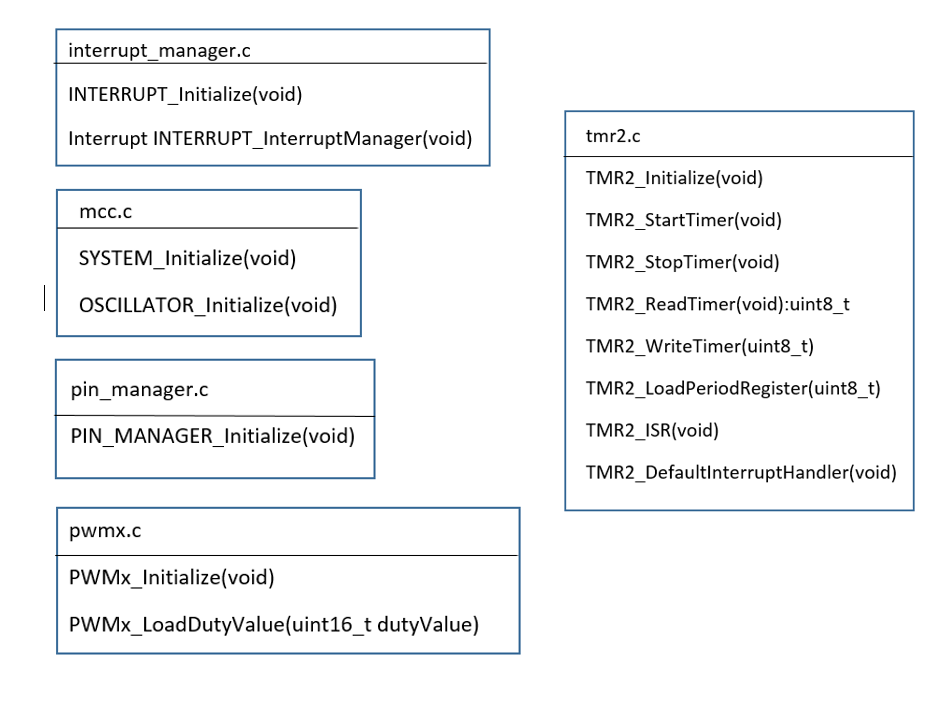


Figure 6 Software Block Digram

3.2.3 Flow Diagram

The software runs based on one 20 ms interrupt. After all registers have been initialized, this interrupt can be raised. When it is raised, the INTERRUPT\_InterruptManager service routine is called. If the correct Timer 2 interrupt bits are set, then the TMR2\_ISR is run. This then calls the function TMR2\_DefaultInterruptHandler. Inside this function is where the PWM settings are handled, specifications of which are discussed in the implementation details (Chapter 4). Once it comes out of this interrupt service routine (ISR), the statements to handle button presses are executed. This complete flow diagram is outlined in Figure 7.

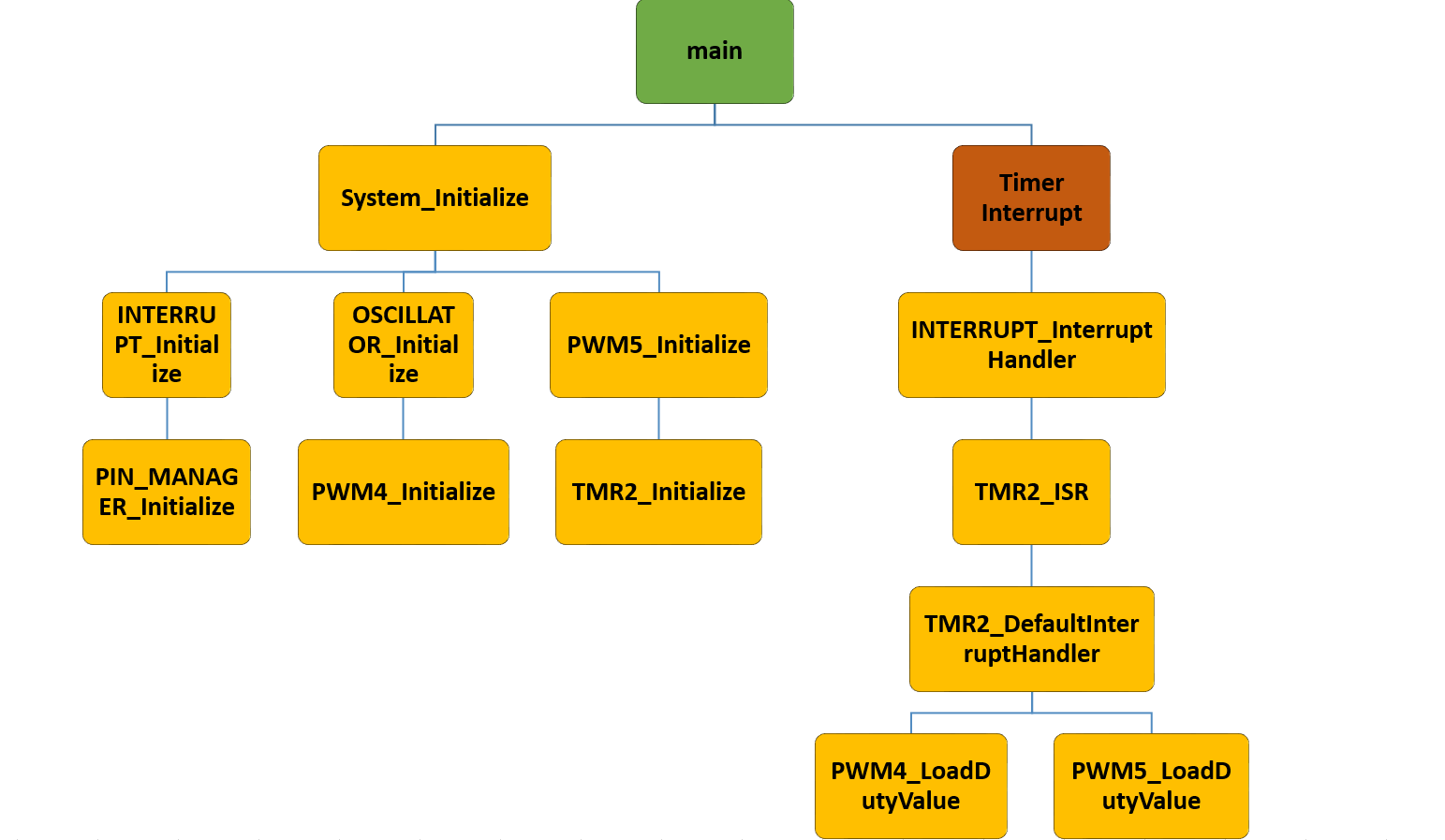


Figure 7 Flow Diagram

CHAPTER 4. IMPLEMENTATION DETAILS

4.1 Parts and Tools

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | QUANTITY | MODEL | SOURCE | PRICE |
| Servo Motor | 2 | RadioShack Standard Servo | Radio Shack | $14.99 |
| Push Buttons | 6 | Panel PCB Momentary Tactile Tact Push Button Switch 4 Pin | Amazon.com | $2.49 (for 100) |
| LEDs | 6 | LED – Assorted (20 pack) | Sparkfun.com | $2.95 (for 20) |
| Transistor | 2 | D44TD3 Silicon NPN Power Transistors | Schweber Electronics | $2.00 |
| 2AA Battery Holder | 1 | N/A | Amazon.com | $3.00 |
| 4 AA Batter Holder | 1 | RadioShack 4AA Batter Holder | RadioShack | $2.49 |
| Jumper Wires | ~40 | Z&T Solderless Breadboard Jumper Wires | Amazon.com | $6.87 (for 100) |
| PIC | 1 | PIC18LF46K22 I/P | Microchipdirect.com | $3.18 |
| Resistors | 10 | 100-piece ½ Watt Resistor Assortment | RadioShack | $7.49 |
| PICkit 3 | 1 | PICkit 3 In-Circuit debugger  PG164130 | Microchipdirect.com | $47.95 |
| TOTAL |  |  |  | $93.41 |

Table 1 Parts and Tools Price List

The end implementation, based on the schematic in Figure 1 can be seen in Figure 8as shown on a bread board. For full pin numbers see Figure 1.

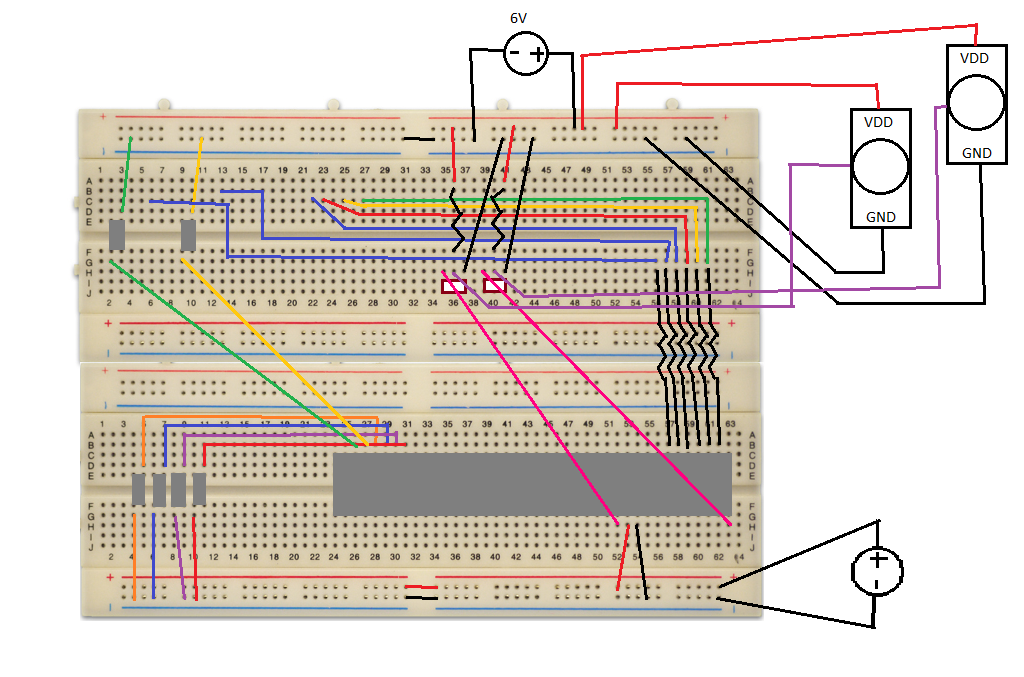


Figure 8 Breadboard Layout of Final Project

Many software aids were used for this project, as outlined in Table 2 below.

|  |  |
| --- | --- |
| **Product** | **Explanation** |
| MPLAB X IDE v3.26 | Integrated Development Environment |
| MPLAB Code Configurator | Used for basic pin setup, plug in proved by MPLAB X IDE |
| X8 Compiler | MPLAB XC Compiler |

Table 2 Software Tools

CHAPTER 5: EXPERIMENT RESULTS

Servo motors usually require a 20 millisecond period. This is calculated according to Equation 1.

Equation 1

Two PWM mode pins were used for this project due to the use of two motors, CCP4 and CCP5, connected to pins RD1 and RE2, respectively. Since the two motors are identical, the same timer, Timer 2, could be attached to them. The Prescaler value of 1:16 was used so the Timer 2 Period could be set to 20 ms. The exact coding values can be seen in Appendix B.

As stated before, both PWM peripherals were attached to the same motor. The reason two PWM peripherals were used, instead of connecting the motors to the same pins, is because the design of the project requires the motors to operate in the same direction, as well as in opposite directions. Within the PWM4 and PWM5 files, there are functions to load the duty value. Per the manual required for this PIC, the duty cycle is set by writing the eight most significant bits of the new duty cycle to the CCPR4 and CCPR5 registers and the two least significant bits of the duty cycle to the CCP4CON and CCP5CON registers.

A servo motor moves to a location on its spectrum (i.e. 0° to 180°) by sending a voltage value for a specified amount of time. For example, sending 6V to the servo motor used for 1.8 ms would result in the motor to rotate to the 120° position. The time chart used for this project is shown in Table 1 below, for a PWM wave operating at 50 Hz.

|  |  |
| --- | --- |
| Duration of Logic High (ms) | Angle (°) |
| 0.6 | 0 |
| 0.9 | 30 |
| 1.2 | 60 |
| 1.5 | 90 |
| 1.8 | 120 |
| 2.1 | 150 |

Table 3 Standard Servo PWM Wave Chart

Because this project only requires the two maximum points to be used, duty cycle values for those two positions were calculated using Equation 2.

Equation 2

For the 0° and 180° positions, the duty cycles calculated are 97% and 89.5%, respectively. These duty cycle percentages are then used in Equation 3 to calculate the value to be entered in the correct registers (CCP4CON, CCP5CON, CCPR4 and CCPR5).

Equation 3

The resulting CCPRxL:CON<5:4> value is used in the PWMx\_LoadDutyValue function. For the 0° position, the function is sent the value 604. For the 180° position, the function is sent the value 585. Figure 9 displays the 97% duty cycle and Figure 10 displays the 89.5% cycle. These photos were taken by connecting an oscilloscope to the output pin of the PIC and the signal wire of the servo motor as the program is running in in-sync mode.

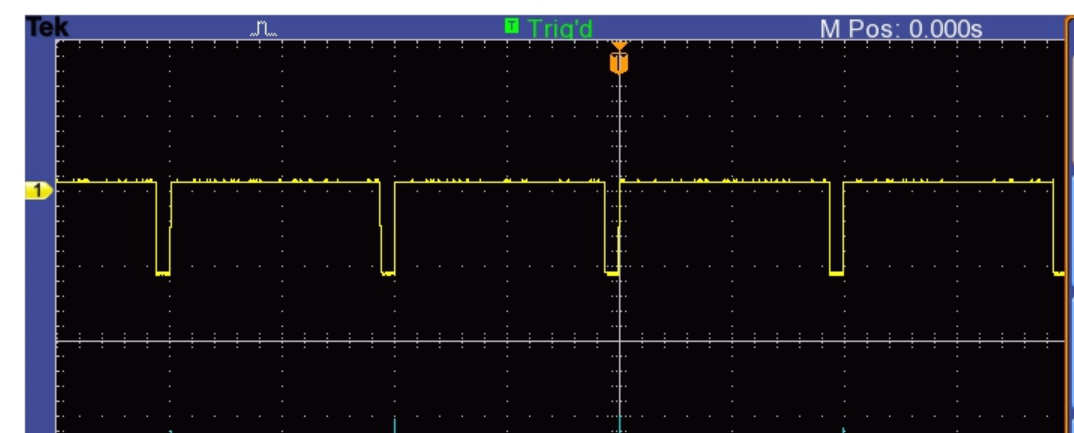


Figure 9 97% Duty Cycle

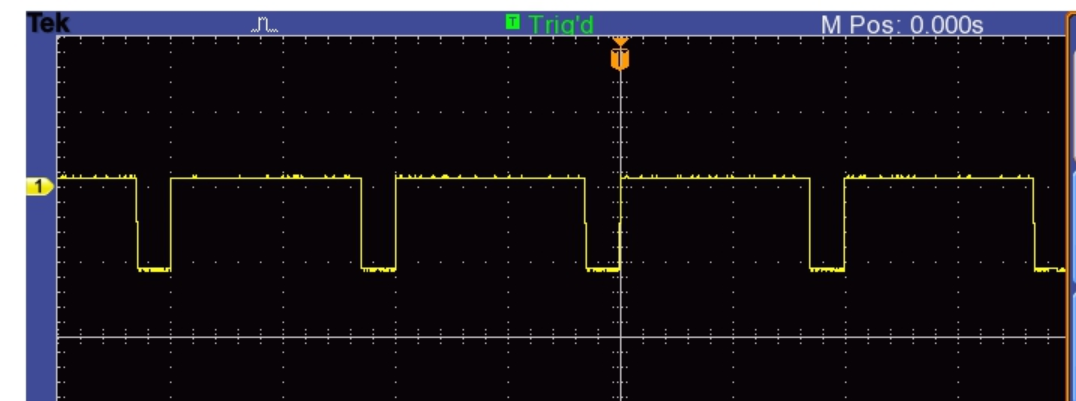


Figure 10 89.5% Duty Cycle

When windshield wipers are operating, they move at the same speed across the windshield, but they have a delay in between position changes to slow down how often they are wiping the window. When this program is just beginning, without speed up, the delay is set to a counter of 50, meaning the motor will change positions every 1000 ms, or 1 sec. This delay counter value is what is changed when the user selects the windshield wipers to speed up or slowdown. The fastest these motors will run is every 400 ms.

The Timer 2 interrupt is enabled to occur every 20 ms to allow a complete PWM period to occur before its duty cycle is changed. If the motors are running, uninterrupted by a speed change, the timer interrupt controls the position change. Every 20 ms, an interrupt is raised, causing the motors to move to the opposite position from where they were at before, but only if the delay counter is reached. If this delay is not reached, the interrupt is exited and the program continues on.

CHAPTER 6: DISCUSSION

When beginning this project, I was not sure what would be feasible to accomplish by myself while completing the necessary requirements. To combat this, I came up with a basic idea to use motors controlled by buttons and uses LEDs. This, of course, needed to serve some sort of purpose. I initially wanted to use stepper motors because I was more comfortable with how they worked, but I was not sure what design to go with, so I bought both servo motors and stepper motors. After troubleshooting some ideas, I settled on windshield wipers. The choice of using a servo motor was natural because of how they operate.

Another major decision was which PIC to use. The PIC 18 is the microcontroller that is suggested, so I exclusively tried those out. I did not go through any major research when picking my initial microcontrollers. I initially bought just 20 pin PIC 18’s. As soon as I received them, I knew there was not going to be enough pins. Before I went to order the 40 pin PICs, I put forth some more research. I found out about the Code Configurator, which only operates with certain PICs. After researching the list, I found a 40 pin module. I chose the Low Power option because I did not realize there was a difference. In the end, this worked out fine because a separate power circuit was needed to operate the servo motors no matter what.

Once I collected all the necessary materials and discovered my project’s purpose, I was able to complete the project with ease.

CHAPTER 7: CONCLUSION

This project gave good insight on how to operate other microcontrollers than the one that we work with in the lab. The Code Configurator provided a simple way to get started on the project organization, so I ran into minimal software issues. Because I am a Computer Engineer, the wiring was difficult. It was necessary to use a transistor, but I had only touched on their use years ago. This meant I needed some research to complete this portion. In the end, the full functionality of the project works quite well with few to no disruptions. This is a success.

REFERENCES

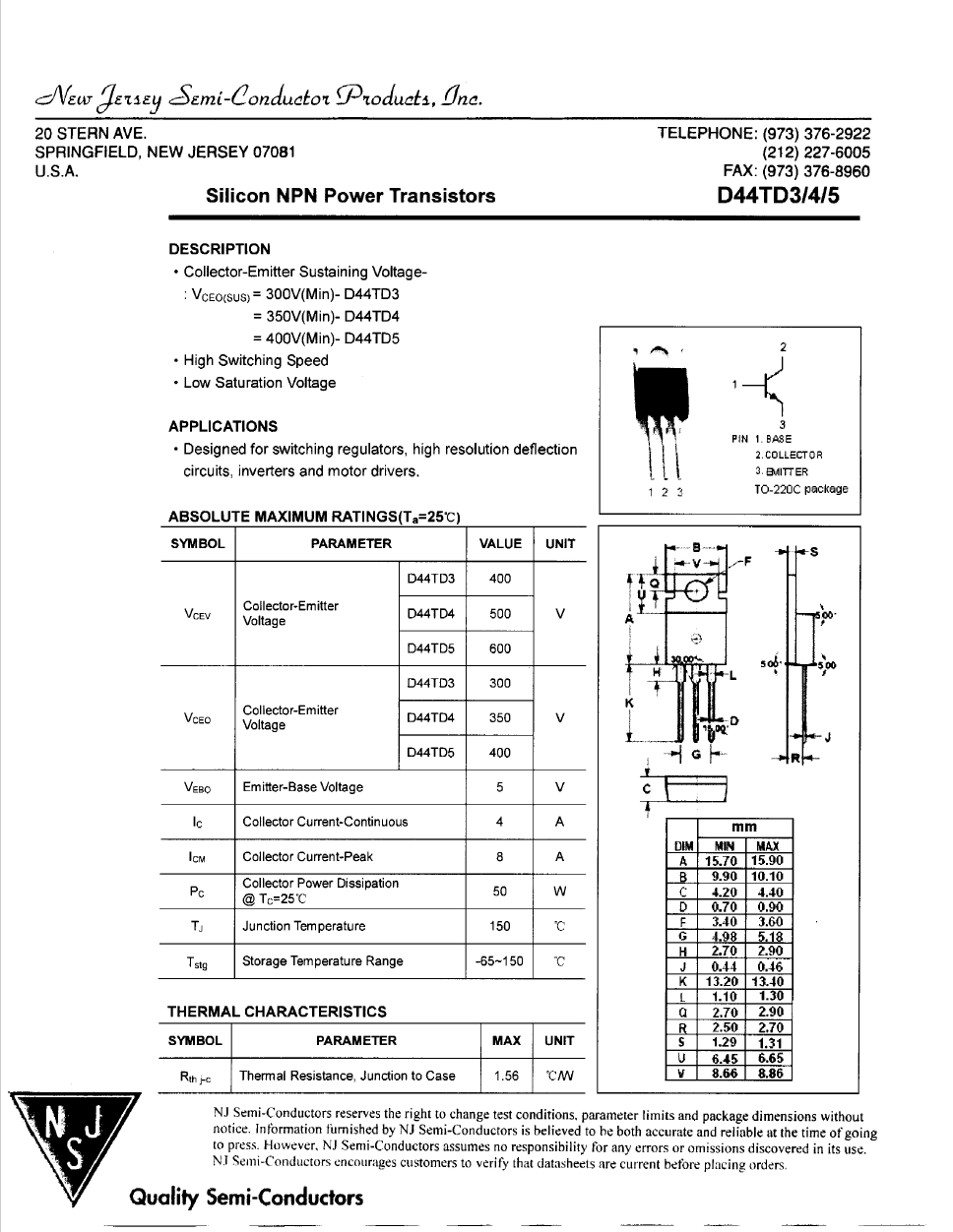
Transistor Information: https://learn.sparkfun.com/tutorials/transistors

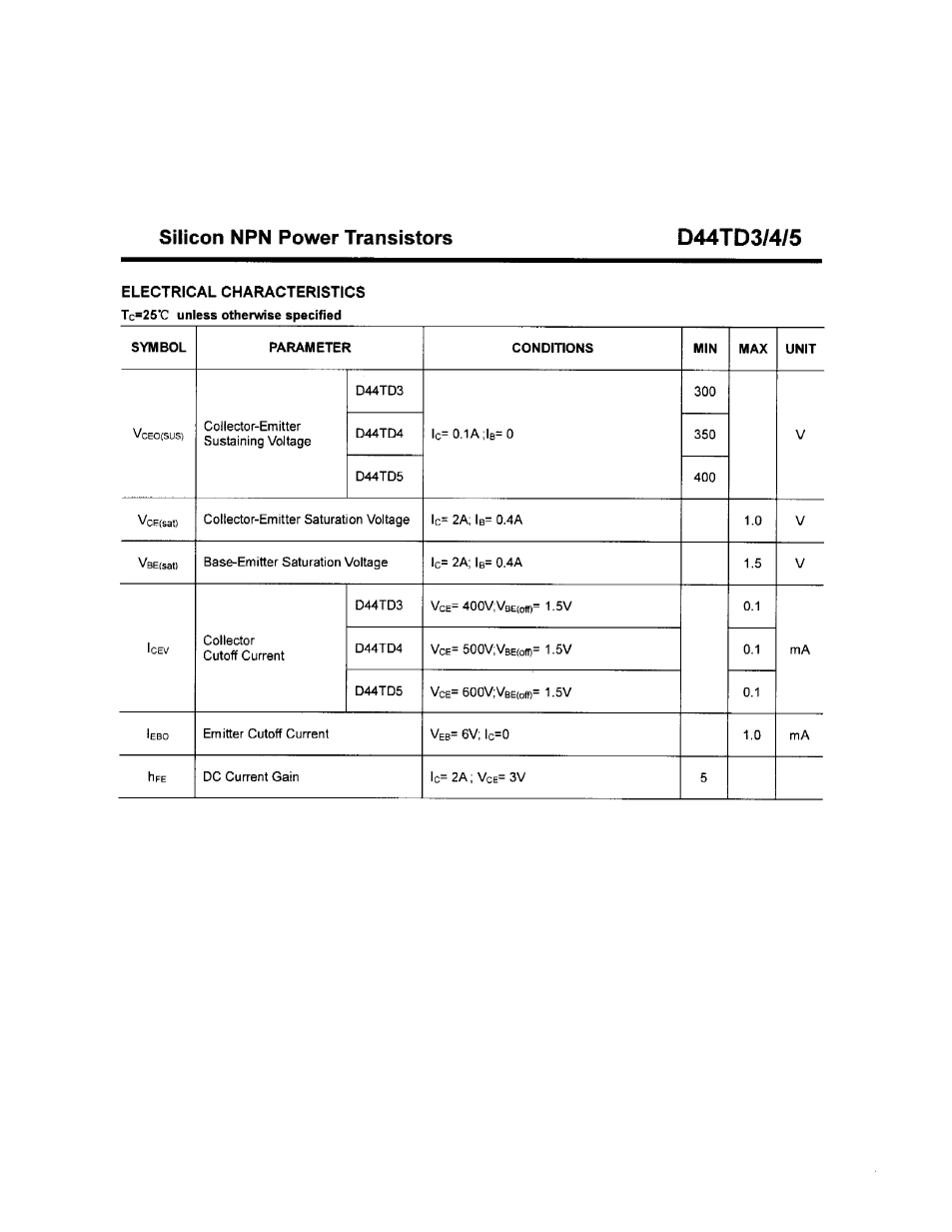
PIC Data Sheet: http://ww1.microchip.com/downloads/en/DeviceDoc/41412F.pdf

Appendices

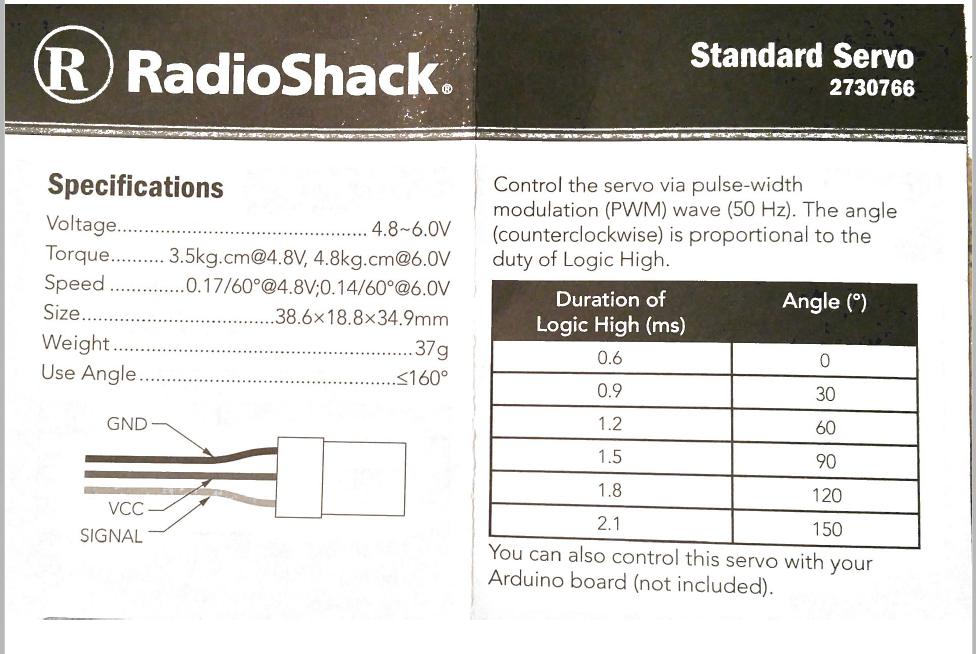
Appendix A: Data Sheets

D44TD3 Transistor





Servo Motor



Appendix B: Code

Main.c

#include "mcc\_generated\_files/mcc.h"

/\*

Main application

\*/

void main(void)

{

// Initialize the device

SYSTEM\_Initialize();

// Enable the Global Interrupts

INTERRUPT\_GlobalInterruptEnable();

//Enable the Peripheral Interrupts

INTERRUPT\_PeripheralInterruptEnable();

while (1)

{

// Add your application code

}

}

Interrupt\_manager.c

#include "interrupt\_manager.h"

#include "mcc.h"

int lightCounter = 0;

int buttonDebounce = 0;

void INTERRUPT\_Initialize (void)

{

// Disable Interrupt Priority Vectors (16CXXX Compatibility Mode)

RCONbits.IPEN = 0;

// Clear peripheral interrupt priority bits (default reset value)

// TMRI

IPR1bits.TMR2IP = 0;

}

void interrupt INTERRUPT\_InterruptManager (void)

{

// interrupt handler

if(PIE1bits.TMR2IE == 1 && PIR1bits.TMR2IF == 1)

{

TMR2\_ISR();

}

else

{

//Unhandled Interrupt

}

lightCounter++;

buttonDebounce++;

if(PORTBbits.RB0 == 0 && !on && buttonDebounce >=10)

{

direction1 = 1;

PWMdelay = 50;

LATD3 = 1;

LATC7 = 1;

on = 1;

buttonDebounce = 0;

}

else if(PORTBbits.RB1 == 0 && !on && buttonDebounce >=10)

{

direction2 = 1;

PWMdelay = 50;

LATD3 = 1;

LATD4 = 1;

on = 1;

buttonDebounce = 0;

}

if(PORTBbits.RB5 == 0)

{

PWMdelay = 50;

LATD3 = 1;

LATC4 = 0;

LATC5 = 0;

LATC6 = 0;

}

if(PORTBbits.RB4 == 0)

{

PWMdelay = 40;

LATD3 = 1;

LATC4 = 1;

LATC5 = 0;

LATC6 = 0;

}

if(PORTBbits.RB3 == 0)

{

PWMdelay = 30;

LATD3 = 1;

LATC4 = 1;

LATC5 = 1;

LATC6 = 0;

}

if(PORTBbits.RB2 == 0)

{

PWMdelay = 20;

LATD3 = 1;

LATC4 = 1;

LATC5 = 1;

LATC6 = 1;

}

if(lightCounter >= 10)

{

if(direction1)

LATC7 = !LATC7;

if(direction2)

LATD4 = !LATD4;

lightCounter=0;

}

}

mcc.c

#pragma config FOSC = INTIO67 // Oscillator Selection bits->Internal oscillator block

#pragma config PLLCFG = OFF // 4X PLL Enable->Oscillator used directly

#pragma config PRICLKEN = ON // Primary clock enable bit->Primary clock is always enabled

#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit->Fail-Safe Clock Monitor disabled

#pragma config IESO = OFF // Internal/External Oscillator Switchover bit->Oscillator Switchover mode disabled

// CONFIG2L

#pragma config PWRTEN = OFF // Power-up Timer Enable bit->Power up timer disabled

#pragma config BOREN = SBORDIS // Brown-out Reset Enable bits->Brown-out Reset enabled in hardware only (SBOREN is disabled)

#pragma config BORV = 190 // Brown Out Reset Voltage bits->VBOR set to 1.90 V nominal

// CONFIG2H

#pragma config WDTEN = OFF // Watchdog Timer Enable bits->Watch dog timer is always disabled. SWDTEN has no effect.

#pragma config WDTPS = 32768 // Watchdog Timer Postscale Select bits->1:32768

// CONFIG3H

#pragma config CCP2MX = PORTC1 // CCP2 MUX bit->CCP2 input/output is multiplexed with RC1

#pragma config PBADEN = ON // PORTB A/D Enable bit->PORTB<5:0> pins are configured as analog input channels on Reset

#pragma config CCP3MX = PORTB5 // P3A/CCP3 Mux bit->P3A/CCP3 input/output is multiplexed with RB5

#pragma config HFOFST = ON // HFINTOSC Fast Start-up->HFINTOSC output and ready status are not delayed by the oscillator stable status

#pragma config T3CMX = PORTC0 // Timer3 Clock input mux bit->T3CKI is on RC0

#pragma config P2BMX = PORTD2 // ECCP2 B output mux bit->P2B is on RD2

#pragma config MCLRE = EXTMCLR // MCLR Pin Enable bit->MCLR pin enabled, RE3 input pin disabled

// CONFIG4L

#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit->Stack full/underflow will cause Reset

#pragma config LVP = OFF // Single-Supply ICSP Enable bit->Single-Supply ICSP disabled

#pragma config XINST = OFF // Extended Instruction Set Enable bit->Instruction set extension and Indexed Addressing mode disabled (Legacy mode)

#pragma config DEBUG = OFF // Background Debug->Disabled

// CONFIG5L

#pragma config CP0 = OFF // Code Protection Block 0->Block 0 (000800-003FFFh) not code-protected

#pragma config CP1 = OFF // Code Protection Block 1->Block 1 (004000-007FFFh) not code-protected

#pragma config CP2 = OFF // Code Protection Block 2->Block 2 (008000-00BFFFh) not code-protected

#pragma config CP3 = OFF // Code Protection Block 3->Block 3 (00C000-00FFFFh) not code-protected

// CONFIG5H

#pragma config CPB = OFF // Boot Block Code Protection bit->Boot block (000000-0007FFh) not code-protected

#pragma config CPD = OFF // Data EEPROM Code Protection bit->Data EEPROM not code-protected

// CONFIG6L

#pragma config WRT0 = OFF // Write Protection Block 0->Block 0 (000800-003FFFh) not write-protected

#pragma config WRT1 = OFF // Write Protection Block 1->Block 1 (004000-007FFFh) not write-protected

#pragma config WRT2 = OFF // Write Protection Block 2->Block 2 (008000-00BFFFh) not write-protected

#pragma config WRT3 = OFF // Write Protection Block 3->Block 3 (00C000-00FFFFh) not write-protected

// CONFIG6H

#pragma config WRTC = OFF // Configuration Register Write Protection bit->Configuration registers (300000-3000FFh) not write-protected

#pragma config WRTB = OFF // Boot Block Write Protection bit->Boot Block (000000-0007FFh) not write-protected

#pragma config WRTD = OFF // Data EEPROM Write Protection bit->Data EEPROM not write-protected

// CONFIG7L

#pragma config EBTR0 = OFF // Table Read Protection Block 0->Block 0 (000800-003FFFh) not protected from table reads executed in other blocks

#pragma config EBTR1 = OFF // Table Read Protection Block 1->Block 1 (004000-007FFFh) not protected from table reads executed in other blocks

#pragma config EBTR2 = OFF // Table Read Protection Block 2->Block 2 (008000-00BFFFh) not protected from table reads executed in other blocks

#pragma config EBTR3 = OFF // Table Read Protection Block 3->Block 3 (00C000-00FFFFh) not protected from table reads executed in other blocks

// CONFIG7H

#pragma config EBTRB = OFF // Boot Block Table Read Protection bit->Boot Block (000000-0007FFh) not protected from table reads executed in other blocks

#include "mcc.h"

void SYSTEM\_Initialize(void)

{

INTERRUPT\_Initialize();

PIN\_MANAGER\_Initialize();

OSCILLATOR\_Initialize();

PWM4\_Initialize();

PWM5\_Initialize();

TMR2\_Initialize();

}

void OSCILLATOR\_Initialize(void)

{

// SCS FOSC; IRCF 1MHz\_HFINTOSC/16; IDLEN disabled;

OSCCON = 0x30;

// PRISD enabled; SOSCGO disabled; MFIOSEL disabled;

OSCCON2 = 0x04;

// INTSRC disabled; PLLEN disabled; TUN 0;

OSCTUNE = 0x00;

// Set the secondary oscillator

}

pin\_manager.c

#include <xc.h>

#include "pin\_manager.h"

void PIN\_MANAGER\_Initialize(void)

{

LATB = 0x0;

LATA = 0x0;

ANSELE = 0x4;

LATE = 0x0;

LATD = 0x0;

LATC = 0x0;

ANSELA = 0x2E;

ANSELB = 0x0;

ANSELC = 0xF8;

ANSELD = 0xFE;

WPUB = 0xFF;

TRISD = 0xE5;

TRISE = 0x3;

TRISB = 0xFF;

TRISC = 0x7;

TRISA = 0xC1;

INTCON2bits.nRBPU = 0x0;

// enable interrupt-on-change globally

// interrupts-on-change are globally disabled

INTCONbits.RBIE = 0;

}

pwm4.c

#include <xc.h>

#include "pwm4.h"

/\*\*

Section: Macro Declarations

\*/

//#define PWM4\_INITIALIZE\_DUTY\_VALUE 529

/\*\*

Section: PWM Module APIs

\*/

void PWM4\_Initialize(void)

{

// Set the PWM to the options selected in the MPLAB(c) Code Configurator

// CCP4M PWM; DC4B 1;

CCP4CON = 0x1C;

// CCPR4L 132;

CCPR4L = 0x84;

// CCPR4H 0;

CCPR4H = 0x00;

// Selecting Timer 2

CCPTMRS1bits.C4TSEL = 0x0;

}

void PWM4\_LoadDutyValue(uint16\_t dutyValue)

{

// Writing to 8 MSBs of pwm duty cycle in CCPRL register

CCPR4L = ((dutyValue & 0x03FC)>>2);

// Writing to 2 LSBs of pwm duty cycle in CCPCON register

CCP4CON = (CCP4CON & 0xCF) | ((dutyValue & 0x0003)<<4);

}

pwm5.c

#include <xc.h>

#include "pwm5.h"

/\*\*

Section: Macro Declarations

\*/

#define PWM5\_INITIALIZE\_DUTY\_VALUE 529

/\*\*

Section: PWM Module APIs

\*/

void PWM5\_Initialize(void)

{

// Set the PWM to the options selected in the MPLAB(c) Code Configurator

// CCP5M PWM; DC5B 1;

CCP5CON = 0x1C;

// CCPR5L 132;

CCPR5L = 0x84;

// CCPR5H 0;

CCPR5H = 0x00;

// Selecting Timer 2

CCPTMRS1bits.C5TSEL = 0x0;

}

void PWM5\_LoadDutyValue(uint16\_t dutyValue)

{

// Writing to 8 MSBs of pwm duty cycle in CCPRL register

CCPR5L = ((dutyValue & 0x03FC)>>2);

// Writing to 2 LSBs of pwm duty cycle in CCPCON register

CCP5CON = (CCP5CON & 0xCF) | ((dutyValue & 0x0003)<<4);

}

tmr2.c

#include <xc.h>

#include "tmr2.h"

#include "pwm4.h"

#include "pwm5.h"

#include "mcc.h"

/\*\*

Section: TMR2 APIs

\*/

int PWMFlag = 0;

int counter = 0;

void TMR2\_Initialize(void)

{

// Set TMR2 to the options selected in the User Interface

// T2CKPS 1:16; T2OUTPS 1:2; TMR2ON off;

T2CON = 0x0A;

// PR2 155;

PR2 = 0x9B;

// TMR2 0;

TMR2 = 0x00;

// Clearing IF flag before enabling the interrupt.

PIR1bits.TMR2IF = 0;

// Enabling TMR2 interrupt.

PIE1bits.TMR2IE = 1;

// Set Default Interrupt Handler

// TMR2\_SetInterruptHandler(TMR2\_DefaultInterruptHandler);

// Start TMR2

TMR2\_StartTimer();

}

void TMR2\_StartTimer(void)

{

// Start the Timer by writing to TMRxON bit

T2CONbits.TMR2ON = 1;

}

void TMR2\_StopTimer(void)

{

// Stop the Timer by writing to TMRxON bit

T2CONbits.TMR2ON = 0;

}

uint8\_t TMR2\_ReadTimer(void)

{

uint8\_t readVal;

readVal = TMR2;

return readVal;

}

void TMR2\_WriteTimer(uint8\_t timerVal)

{

// Write to the Timer2 register

TMR2 = timerVal;

}

void TMR2\_LoadPeriodRegister(uint8\_t periodVal)

{

PR2 = periodVal;

}

void TMR2\_ISR(void)

{

// clear the TMR2 interrupt flag

PIR1bits.TMR2IF = 0;

// ticker function call;

// ticker is 1 -> Callback function gets called everytime this ISR executes

//TMR2\_CallBack();

TMR2\_DefaultInterruptHandler();

}

void TMR2\_CallBack(void)

{

// Add your custom callback code here

// this code executes every TMR2\_INTERRUPT\_TICKER\_FACTOR periods of TMR2

if(TMR2\_InterruptHandler)

{

TMR2\_InterruptHandler();

}

}

void TMR2\_SetInterruptHandler(void\* InterruptHandler){

TMR2\_InterruptHandler = InterruptHandler;

}

void TMR2\_DefaultInterruptHandler(void){

// add your TMR2 interrupt custom code

// or set custom function using TMR2\_SetInterruptHandler()

if(on)

{//direction = 1 then motors move together

if(direction1 == 1)

{

if(counter>=PWMdelay)

{

if(PWMFlag == 1)

{

PWM4\_LoadDutyValue(529);

PWM5\_LoadDutyValue(529);

PWMFlag = 0;

}

else

{

PWM4\_LoadDutyValue(585);

PWM5\_LoadDutyValue(585);//604

PWMFlag = 1;

}

counter = 0;

}

else

counter++;

}

else if(direction2 == 1)

{

if(counter>=PWMdelay)

{

if(PWMFlag == 1)

{

PWM4\_LoadDutyValue(529);

PWM5\_LoadDutyValue(585);

PWMFlag = 0;

}

else

{

PWM4\_LoadDutyValue(585);

PWM5\_LoadDutyValue(529);

PWMFlag = 1;

}

counter = 0;

}

else

counter++;

}

}

}

Interrupt\_manager.h

#ifndef INTERRUPT\_MANAGER\_H

#define INTERRUPT\_MANAGER\_H

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will enable global interrupts.

\* @Example

INTERRUPT\_GlobalInterruptEnable();

\*/

#define INTERRUPT\_GlobalInterruptEnable() (INTCONbits.GIE = 1)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will disable global interrupts.

\* @Example

INTERRUPT\_GlobalInterruptDisable();

\*/

#define INTERRUPT\_GlobalInterruptDisable() (INTCONbits.GIE = 0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will enable peripheral interrupts.

\* @Example

INTERRUPT\_PeripheralInterruptEnable();

\*/

#define INTERRUPT\_PeripheralInterruptEnable() (INTCONbits.PEIE = 1)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will disable peripheral interrupts.

\* @Example

INTERRUPT\_PeripheralInterruptDisable();

\*/

#define INTERRUPT\_PeripheralInterruptDisable() (INTCONbits.PEIE = 0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes PIC18 peripheral interrupt priorities; enables/disables priority vectors

\* @Example

INTERRUPT\_Initialize();

\*/

void INTERRUPT\_Initialize (void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Interrupt service routine. Calls module interrupt handlers.

\* @Example

INTERRUPT\_InterruptManager();

\*/

void interrupt INTERRUPT\_InterruptManager (void);

#endif // INTERRUPT\_MANAGER\_H

mcc.h

#ifndef MCC\_H

#define MCC\_H

#include <xc.h>

#include "pin\_manager.h"

#include <stdint.h>

#include <stdbool.h>

#include "interrupt\_manager.h"

#include "pwm4.h"

#include "pwm5.h"

#include "tmr2.h"

#define \_XTAL\_FREQ 1000000

int PWMdelay = 0;

int direction1 = 0;

int direction2 = 0;

int on = 0;

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes the device to the default states configured in the

\* MCC GUI

\* @Example

SYSTEM\_Initialize(void);

\*/

void SYSTEM\_Initialize(void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes the oscillator to the default states configured in the

\* MCC GUI

\* @Example

OSCILLATOR\_Initialize(void);

\*/

void OSCILLATOR\_Initialize(void);

#endif /\* MCC\_H \*/

Pin\_manager.h

#ifndef PIN\_MANAGER\_H

#define PIN\_MANAGER\_H

#define INPUT 1

#define OUTPUT 0

#define HIGH 1

#define LOW 0

#define ANALOG 1

#define DIGITAL 0

#define PULL\_UP\_ENABLED 1

#define PULL\_UP\_DISABLED 0

// get/set IO\_RB0 aliases

#define IO\_RB0\_TRIS TRISB0

#define IO\_RB0\_LAT LATB0

#define IO\_RB0\_PORT RB0

#define IO\_RB0\_WPU WPUB0

#define IO\_RB0\_ANS ANSB0

#define IO\_RB0\_SetHigh() do { LATB0 = 1; } while(0)

#define IO\_RB0\_SetLow() do { LATB0 = 0; } while(0)

#define IO\_RB0\_Toggle() do { LATB0 = ~LATB0; } while(0)

#define IO\_RB0\_GetValue() RB0

#define IO\_RB0\_SetDigitalInput() do { TRISB0 = 1; } while(0)

#define IO\_RB0\_SetDigitalOutput() do { TRISB0 = 0; } while(0)

#define IO\_RB0\_SetPullup() do { WPUB0 = 1; } while(0)

#define IO\_RB0\_ResetPullup() do { WPUB0 = 0; } while(0)

#define IO\_RB0\_SetAnalogMode() do { ANSB0 = 1; } while(0)

#define IO\_RB0\_SetDigitalMode() do { ANSB0 = 0; } while(0)

// get/set IO\_RB1 aliases

#define IO\_RB1\_TRIS TRISB1

#define IO\_RB1\_LAT LATB1

#define IO\_RB1\_PORT RB1

#define IO\_RB1\_WPU WPUB1

#define IO\_RB1\_ANS ANSB1

#define IO\_RB1\_SetHigh() do { LATB1 = 1; } while(0)

#define IO\_RB1\_SetLow() do { LATB1 = 0; } while(0)

#define IO\_RB1\_Toggle() do { LATB1 = ~LATB1; } while(0)

#define IO\_RB1\_GetValue() RB1

#define IO\_RB1\_SetDigitalInput() do { TRISB1 = 1; } while(0)

#define IO\_RB1\_SetDigitalOutput() do { TRISB1 = 0; } while(0)

#define IO\_RB1\_SetPullup() do { WPUB1 = 1; } while(0)

#define IO\_RB1\_ResetPullup() do { WPUB1 = 0; } while(0)

#define IO\_RB1\_SetAnalogMode() do { ANSB1 = 1; } while(0)

#define IO\_RB1\_SetDigitalMode() do { ANSB1 = 0; } while(0)

// get/set IO\_RB2 aliases

#define IO\_RB2\_TRIS TRISB2

#define IO\_RB2\_LAT LATB2

#define IO\_RB2\_PORT RB2

#define IO\_RB2\_WPU WPUB2

#define IO\_RB2\_ANS ANSB2

#define IO\_RB2\_SetHigh() do { LATB2 = 1; } while(0)

#define IO\_RB2\_SetLow() do { LATB2 = 0; } while(0)

#define IO\_RB2\_Toggle() do { LATB2 = ~LATB2; } while(0)

#define IO\_RB2\_GetValue() RB2

#define IO\_RB2\_SetDigitalInput() do { TRISB2 = 1; } while(0)

#define IO\_RB2\_SetDigitalOutput() do { TRISB2 = 0; } while(0)

#define IO\_RB2\_SetPullup() do { WPUB2 = 1; } while(0)

#define IO\_RB2\_ResetPullup() do { WPUB2 = 0; } while(0)

#define IO\_RB2\_SetAnalogMode() do { ANSB2 = 1; } while(0)

#define IO\_RB2\_SetDigitalMode() do { ANSB2 = 0; } while(0)

// get/set IO\_RB3 aliases

#define IO\_RB3\_TRIS TRISB3

#define IO\_RB3\_LAT LATB3

#define IO\_RB3\_PORT RB3

#define IO\_RB3\_WPU WPUB3

#define IO\_RB3\_ANS ANSB3

#define IO\_RB3\_SetHigh() do { LATB3 = 1; } while(0)

#define IO\_RB3\_SetLow() do { LATB3 = 0; } while(0)

#define IO\_RB3\_Toggle() do { LATB3 = ~LATB3; } while(0)

#define IO\_RB3\_GetValue() RB3

#define IO\_RB3\_SetDigitalInput() do { TRISB3 = 1; } while(0)

#define IO\_RB3\_SetDigitalOutput() do { TRISB3 = 0; } while(0)

#define IO\_RB3\_SetPullup() do { WPUB3 = 1; } while(0)

#define IO\_RB3\_ResetPullup() do { WPUB3 = 0; } while(0)

#define IO\_RB3\_SetAnalogMode() do { ANSB3 = 1; } while(0)

#define IO\_RB3\_SetDigitalMode() do { ANSB3 = 0; } while(0)

// get/set IO\_RB4 aliases

#define IO\_RB4\_TRIS TRISB4

#define IO\_RB4\_LAT LATB4

#define IO\_RB4\_PORT RB4

#define IO\_RB4\_WPU WPUB4

#define IO\_RB4\_ANS ANSB4

#define IO\_RB4\_SetHigh() do { LATB4 = 1; } while(0)

#define IO\_RB4\_SetLow() do { LATB4 = 0; } while(0)

#define IO\_RB4\_Toggle() do { LATB4 = ~LATB4; } while(0)

#define IO\_RB4\_GetValue() RB4

#define IO\_RB4\_SetDigitalInput() do { TRISB4 = 1; } while(0)

#define IO\_RB4\_SetDigitalOutput() do { TRISB4 = 0; } while(0)

#define IO\_RB4\_SetPullup() do { WPUB4 = 1; } while(0)

#define IO\_RB4\_ResetPullup() do { WPUB4 = 0; } while(0)

#define IO\_RB4\_SetAnalogMode() do { ANSB4 = 1; } while(0)

#define IO\_RB4\_SetDigitalMode() do { ANSB4 = 0; } while(0)

// get/set IO\_RB5 aliases

#define IO\_RB5\_TRIS TRISB5

#define IO\_RB5\_LAT LATB5

#define IO\_RB5\_PORT RB5

#define IO\_RB5\_WPU WPUB5

#define IO\_RB5\_ANS ANSB5

#define IO\_RB5\_SetHigh() do { LATB5 = 1; } while(0)

#define IO\_RB5\_SetLow() do { LATB5 = 0; } while(0)

#define IO\_RB5\_Toggle() do { LATB5 = ~LATB5; } while(0)

#define IO\_RB5\_GetValue() RB5

#define IO\_RB5\_SetDigitalInput() do { TRISB5 = 1; } while(0)

#define IO\_RB5\_SetDigitalOutput() do { TRISB5 = 0; } while(0)

#define IO\_RB5\_SetPullup() do { WPUB5 = 1; } while(0)

#define IO\_RB5\_ResetPullup() do { WPUB5 = 0; } while(0)

#define IO\_RB5\_SetAnalogMode() do { ANSB5 = 1; } while(0)

#define IO\_RB5\_SetDigitalMode() do { ANSB5 = 0; } while(0)

// get/set IO\_RC3 aliases

#define IO\_RC3\_TRIS TRISC3

#define IO\_RC3\_LAT LATC3

#define IO\_RC3\_PORT RC3

#define IO\_RC3\_ANS ANSC3

#define IO\_RC3\_SetHigh() do { LATC3 = 1; } while(0)

#define IO\_RC3\_SetLow() do { LATC3 = 0; } while(0)

#define IO\_RC3\_Toggle() do { LATC3 = ~LATC3; } while(0)

#define IO\_RC3\_GetValue() RC3

#define IO\_RC3\_SetDigitalInput() do { TRISC3 = 1; } while(0)

#define IO\_RC3\_SetDigitalOutput() do { TRISC3 = 0; } while(0)

#define IO\_RC3\_SetAnalogMode() do { ANSC3 = 1; } while(0)

#define IO\_RC3\_SetDigitalMode() do { ANSC3 = 0; } while(0)

// get/set IO\_RC4 aliases

#define IO\_RC4\_TRIS TRISC4

#define IO\_RC4\_LAT LATC4

#define IO\_RC4\_PORT RC4

#define IO\_RC4\_ANS ANSC4

#define IO\_RC4\_SetHigh() do { LATC4 = 1; } while(0)

#define IO\_RC4\_SetLow() do { LATC4 = 0; } while(0)

#define IO\_RC4\_Toggle() do { LATC4 = ~LATC4; } while(0)

#define IO\_RC4\_GetValue() RC4

#define IO\_RC4\_SetDigitalInput() do { TRISC4 = 1; } while(0)

#define IO\_RC4\_SetDigitalOutput() do { TRISC4 = 0; } while(0)

#define IO\_RC4\_SetAnalogMode() do { ANSC4 = 1; } while(0)

#define IO\_RC4\_SetDigitalMode() do { ANSC4 = 0; } while(0)

// get/set IO\_RC5 aliases

#define IO\_RC5\_TRIS TRISC5

#define IO\_RC5\_LAT LATC5

#define IO\_RC5\_PORT RC5

#define IO\_RC5\_ANS ANSC5

#define IO\_RC5\_SetHigh() do { LATC5 = 1; } while(0)

#define IO\_RC5\_SetLow() do { LATC5 = 0; } while(0)

#define IO\_RC5\_Toggle() do { LATC5 = ~LATC5; } while(0)

#define IO\_RC5\_GetValue() RC5

#define IO\_RC5\_SetDigitalInput() do { TRISC5 = 1; } while(0)

#define IO\_RC5\_SetDigitalOutput() do { TRISC5 = 0; } while(0)

#define IO\_RC5\_SetAnalogMode() do { ANSC5 = 1; } while(0)

#define IO\_RC5\_SetDigitalMode() do { ANSC5 = 0; } while(0)

// get/set IO\_RC6 aliases

#define IO\_RC6\_TRIS TRISC6

#define IO\_RC6\_LAT LATC6

#define IO\_RC6\_PORT RC6

#define IO\_RC6\_ANS ANSC6

#define IO\_RC6\_SetHigh() do { LATC6 = 1; } while(0)

#define IO\_RC6\_SetLow() do { LATC6 = 0; } while(0)

#define IO\_RC6\_Toggle() do { LATC6 = ~LATC6; } while(0)

#define IO\_RC6\_GetValue() RC6

#define IO\_RC6\_SetDigitalInput() do { TRISC6 = 1; } while(0)

#define IO\_RC6\_SetDigitalOutput() do { TRISC6 = 0; } while(0)

#define IO\_RC6\_SetAnalogMode() do { ANSC6 = 1; } while(0)

#define IO\_RC6\_SetDigitalMode() do { ANSC6 = 0; } while(0)

// get/set IO\_RC7 aliases

#define IO\_RC7\_TRIS TRISC7

#define IO\_RC7\_LAT LATC7

#define IO\_RC7\_PORT RC7

#define IO\_RC7\_ANS ANSC7

#define IO\_RC7\_SetHigh() do { LATC7 = 1; } while(0)

#define IO\_RC7\_SetLow() do { LATC7 = 0; } while(0)

#define IO\_RC7\_Toggle() do { LATC7 = ~LATC7; } while(0)

#define IO\_RC7\_GetValue() RC7

#define IO\_RC7\_SetDigitalInput() do { TRISC7 = 1; } while(0)

#define IO\_RC7\_SetDigitalOutput() do { TRISC7 = 0; } while(0)

#define IO\_RC7\_SetAnalogMode() do { ANSC7 = 1; } while(0)

#define IO\_RC7\_SetDigitalMode() do { ANSC7 = 0; } while(0)

// get/set IO\_RC7 aliases

#define IO\_RC7\_TRIS TRISC7

#define IO\_RC7\_LAT LATC7

#define IO\_RC7\_PORT RC7

#define IO\_RC7\_ANS ANSC7

#define IO\_RC7\_SetHigh() do { LATC7 = 1; } while(0)

#define IO\_RC7\_SetLow() do { LATC7 = 0; } while(0)

#define IO\_RC7\_Toggle() do { LATC7 = ~LATC7; } while(0)

#define IO\_RC7\_GetValue() RC7

#define IO\_RC7\_SetDigitalInput() do { TRISC7 = 1; } while(0)

#define IO\_RC7\_SetDigitalOutput() do { TRISC7 = 0; } while(0)

#define IO\_RC7\_SetAnalogMode() do { ANSC7 = 1; } while(0)

#define IO\_RC7\_SetDigitalMode() do { ANSC7 = 0; } while(0)

// get/set IO\_RD3 aliases

#define IO\_RD3\_TRIS TRISD3

#define IO\_RD3\_LAT LATD3

#define IO\_RD3\_PORT RD3

#define IO\_RD3\_ANS ANSD3

#define IO\_RD3\_SetHigh() do { LATD3 = 1; } while(0)

#define IO\_RD3\_SetLow() do { LATD3 = 0; } while(0)

#define IO\_RD3\_Toggle() do { LATD3 = ~LATD3; } while(0)

#define IO\_RD3\_GetValue() RD3

#define IO\_RD3\_SetDigitalInput() do { TRISD3 = 1; } while(0)

#define IO\_RD3\_SetDigitalOutput() do { TRISD3 = 0; } while(0)

#define IO\_RD3\_SetAnalogMode() do { ANSD3 = 1; } while(0)

#define IO\_RD3\_SetDigitalMode() do { ANSD3 = 0; } while(0)

// get/set IO\_RD4 aliases

#define IO\_RD4\_TRIS TRISD4

#define IO\_RD4\_LAT LATD4

#define IO\_RD4\_PORT RD4

#define IO\_RD4\_ANS ANSD4

#define IO\_RD4\_SetHigh() do { LATD4 = 1; } while(0)

#define IO\_RD4\_SetLow() do { LATD4 = 0; } while(0)

#define IO\_RD4\_Toggle() do { LATD4 = ~LATD4; } while(0)

#define IO\_RD4\_GetValue() RD4

#define IO\_RD4\_SetDigitalInput() do { TRISD4 = 1; } while(0)

#define IO\_RD4\_SetDigitalOutput() do { TRISD4 = 0; } while(0)

#define IO\_RD4\_SetAnalogMode() do { ANSD4 = 1; } while(0)

#define IO\_RD4\_SetDigitalMode() do { ANSD4 = 0; } while(0)

// get/set IO\_RD4 aliases

#define IO\_RD4\_TRIS TRISD4

#define IO\_RD4\_LAT LATD4

#define IO\_RD4\_PORT RD4

#define IO\_RD4\_ANS ANSD4

#define IO\_RD4\_SetHigh() do { LATD4 = 1; } while(0)

#define IO\_RD4\_SetLow() do { LATD4 = 0; } while(0)

#define IO\_RD4\_Toggle() do { LATD4 = ~LATD4; } while(0)

#define IO\_RD4\_GetValue() RD4

#define IO\_RD4\_SetDigitalInput() do { TRISD4 = 1; } while(0)

#define IO\_RD4\_SetDigitalOutput() do { TRISD4 = 0; } while(0)

#define IO\_RD4\_SetAnalogMode() do { ANSD4 = 1; } while(0)

#define IO\_RD4\_SetDigitalMode() do { ANSD4 = 0; } while(0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

GPIO and peripheral I/O initialization

\* @Example

PIN\_MANAGER\_Initialize();

\*/

void PIN\_MANAGER\_Initialize (void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Interrupt on Change Handling routine

\* @Example

PIN\_MANAGER\_IOC();

\*/

pwm4.h

#ifndef \_PWM4\_H

#define \_PWM4\_H

/\*\*

Section: Included Files

\*/

#include <xc.h>

#include <stdint.h>

#include <stdbool.h>

#ifdef \_\_cplusplus // Provide C++ Compatibility

extern "C" {

#endif

/\*\*

Section: PWM Module APIs

\*/

/\*\*

@Summary

Initializes the CCP4

@Description

This routine initializes the CCP4 module.

This routine must be called before any other CCP4 routine is called.

This routine should only be called once during system initialization.

@Preconditions

None

@Param

None

@Returns

None

@Comment

@Example

<code>

uint16\_t dutycycle;

PWM4\_Initialize();

PWM4\_LoadDutyValue(dutycycle);

</code>

\*/

void PWM4\_Initialize(void);

/\*\*

@Summary

Loads 16-bit duty cycle.

@Description

This routine loads the 16 bit duty cycle value.

@Preconditions

PWM4\_Initialize() function should have been called

before calling this function.

@Param

Pass 16bit duty cycle value.

@Returns

None

@Example

<code>

uint16\_t dutycycle;

PWM4\_Initialize();

PWM4\_LoadDutyValue(dutycycle);

</code>

\*/

void PWM4\_LoadDutyValue(uint16\_t dutyValue);

#ifdef \_\_cplusplus // Provide C++ Compatibility

}

#endif

#endif //PWM4\_H

pwm5.h

#include <xc.h>

#include <stdint.h>

#include <stdbool.h>

#ifdef \_\_cplusplus // Provide C++ Compatibility

extern "C" {

#endif

/\*\*

Section: PWM Module APIs

\*/

/\*\*

@Summary

Initializes the CCP5

@Description

This routine initializes the CCP5 module.

This routine must be called before any other CCP5 routine is called.

This routine should only be called once during system initialization.

@Preconditions

None

@Param

None

@Returns

None

@Comment

@Example

<code>

uint16\_t dutycycle;

PWM5\_Initialize();

PWM5\_LoadDutyValue(dutycycle);

</code>

\*/

void PWM5\_Initialize(void);

/\*\*

@Summary

Loads 16-bit duty cycle.

@Description

This routine loads the 16 bit duty cycle value.

@Preconditions

PWM5\_Initialize() function should have been called

before calling this function.

@Param

Pass 16bit duty cycle value.

@Returns

None

@Example

<code>

uint16\_t dutycycle;

PWM5\_Initialize();

PWM5\_LoadDutyValue(dutycycle);

</code>

\*/

void PWM5\_LoadDutyValue(uint16\_t dutyValue);

#ifdef \_\_cplusplus // Provide C++ Compatibility

}

#endif

#endif //PWM5\_H

/\*\*

tmr2.h

#ifndef \_TMR2\_H

#define \_TMR2\_H

/\*\*

Section: Included Files

\*/

#include <stdint.h>

#include <stdbool.h>

#ifdef \_\_cplusplus // Provide C++ Compatibility

extern "C" {

#endif

/\*\*

Section: Macro Declarations

\*/

/\*\*

Section: TMR2 APIs

\*/

/\*\*

@Summary

Initializes the TMR2 module.

@Description

This function initializes the TMR2 Registers.

This function must be called before any other TMR2 function is called.

@Preconditions

None

@Param

None

@Returns

None

@Comment

@Example

<code>

main()

{

// Initialize TMR2 module

TMR2\_Initialize();

// Do something else...

}

</code>

\*/

void TMR2\_Initialize(void);

/\*\*

@Summary

This function starts the TMR2.

@Description

This function starts the TMR2 operation.

This function must be called after the initialization of TMR2.

@Preconditions

Initialize the TMR2 before calling this function.

@Param

None

@Returns

None

@Example

<code>

// Initialize TMR2 module

// Start TMR2

TMR2\_StartTimer();

// Do something else...

</code>

\*/

void TMR2\_StartTimer(void);

/\*\*

@Summary

This function stops the TMR2.

@Description

This function stops the TMR2 operation.

This function must be called after the start of TMR2.

@Preconditions

Initialize the TMR2 before calling this function.

@Param

None

@Returns

None

@Example

<code>

// Initialize TMR2 module

// Start TMR2

TMR2\_StartTimer();

// Do something else...

// Stop TMR2;

TMR2\_StopTimer();

</code>

\*/

void TMR2\_StopTimer(void);

/\*\*

@Summary

Reads the TMR2 register.

@Description

This function reads the TMR2 register value and return it.

@Preconditions

Initialize the TMR2 before calling this function.

@Param

None

@Returns

This function returns the current value of TMR2 register.

@Example

<code>

// Initialize TMR2 module

// Start TMR2

TMR2\_StartTimer();

// Read the current value of TMR2

if(0 == TMR2\_ReadTimer())

{

// Do something else...

// Reload the TMR value

TMR2\_Reload();

}

</code>

\*/

uint8\_t TMR2\_ReadTimer(void);

/\*\*

@Summary

Writes the TMR2 register.

@Description

This function writes the TMR2 register.

This function must be called after the initialization of TMR2.

@Preconditions

Initialize the TMR2 before calling this function.

@Param

timerVal - Value to write into TMR2 register.

@Returns

None

@Example

<code>

#define PERIOD 0x80

#define ZERO 0x00

while(1)

{

// Read the TMR2 register

if(ZERO == TMR2\_ReadTimer())

{

// Do something else...

// Write the TMR2 register

TMR2\_WriteTimer(PERIOD);

}

// Do something else...

}

</code>

\*/

void TMR2\_WriteTimer(uint8\_t timerVal);

/\*\*

@Summary

Load value to Period Register.

@Description

This function writes the value to PR2 register.

This function must be called after the initialization of TMR2.

@Preconditions

Initialize the TMR2 before calling this function.

@Param

periodVal - Value to load into TMR2 register.

@Returns

None

@Example

<code>

#define PERIOD1 0x80

#define PERIOD2 0x40

#define ZERO 0x00

while(1)

{

// Read the TMR2 register

if(ZERO == TMR2\_ReadTimer())

{

// Do something else...

if(flag)

{

flag = 0;

// Load Period 1 value

TMR2\_LoadPeriodRegister(PERIOD1);

}

else

{

flag = 1;

// Load Period 2 value

TMR2\_LoadPeriodRegister(PERIOD2);

}

}

// Do something else...

}

</code>

\*/

void TMR2\_LoadPeriodRegister(uint8\_t periodVal);

/\*\*

@Summary

Timer Interrupt Service Routine

@Description

Timer Interrupt Service Routine is called by the Interrupt Manager.

@Preconditions

Initialize the TMR2 module with interrupt before calling this isr.

@Param

None

@Returns

None

\*/

void TMR2\_ISR(void);

/\*\*

@Summary

CallBack function

@Description

This function is called from the timer ISR. User can write your code in this function.

@Preconditions

Initialize the TMR2 module with interrupt before calling this function.

@Param

None

@Returns

None

\*/

void TMR2\_CallBack(void);

/\*\*

@Summary

Set Timer Interrupt Handler

@Description

This sets the function to be called during the ISR

@Preconditions

Initialize the TMR2 module with interrupt before calling this.

@Param

Address of function to be set

@Returns

None

\*/

void TMR2\_SetInterruptHandler(void \*InterruptHandler);

/\*\*

@Summary

Timer Interrupt Handler

@Description

This is a function pointer to the function that will be called during the ISR

@Preconditions

Initialize the TMR2 module with interrupt before calling this isr.

@Param

None

@Returns

None

\*/

void (\*TMR2\_InterruptHandler)(void);

/\*\*

@Summary

Default Timer Interrupt Handler

@Description

This is the default Interrupt Handler function

@Preconditions

Initialize the TMR2 module with interrupt before calling this isr.

@Param

None

@Returns

None

\*/

void TMR2\_DefaultInterruptHandler(void);

#ifdef \_\_cplusplus // Provide C++ Compatibility

}

#endif

#endif // \_TMR2\_H