CECS 347 Spring 2021 Project 1

Robot Car with Motor Control

By

Len Quach

03/12/2021

Build a wheeled robot that changes its

speed and direction according to the input switches of the TM4C123G LaunchPad

Microcontroller.

Introduction

The purpose of this project is to practice how to use hardware PWM and PLL hardware components, and review GPIO and interrupts. We are building a wheel robot that changes its speed and direction according to the input switches of the TM4C123G LaunchPad Microcontroller.

Switch 1 will be used to increase the speed of the robot with switch debounce. Robot will start with no motion, press once will put the robot in starting speed, around 30% duty cycle, keep pressing the robot speed will go through 60%, 80%, 98%, stop and the cycle repeats. We will use hardware PWM for speed control. Next, switch 2 will be used to control moving direction of the robot with switch debounce. Robot will start in forward direction mode, press once the robot will be in backward direction mode, press again and the cycle repeats.

There are 3 LEDs outputs will be used in this project including RED which is indicates no robot motion, BLUE indicates backward direction, and GREEN indicates forward direction. For hardware PWM implementation, we will need any two hardware PWM outputs available on TM4C123 Launchpad to drive the two DC motors. 50MHz system clock will be generated using PLL.

Operation

https://youtu.be/ZcHU34E6DG4

Theory

Starting from Lab2, I’m using 2 PWM outputs for two motors which is PB6 and PB7. Testing speed control with the two push buttons on the Launchpad and two LEDs on breadboard. Modify function GPIOPortF_Handler() SW1 touch logic to cycle through stop, 30%, 60%, 80% and 98% speed. Then test PWM outputs with Analog Discovery 2.

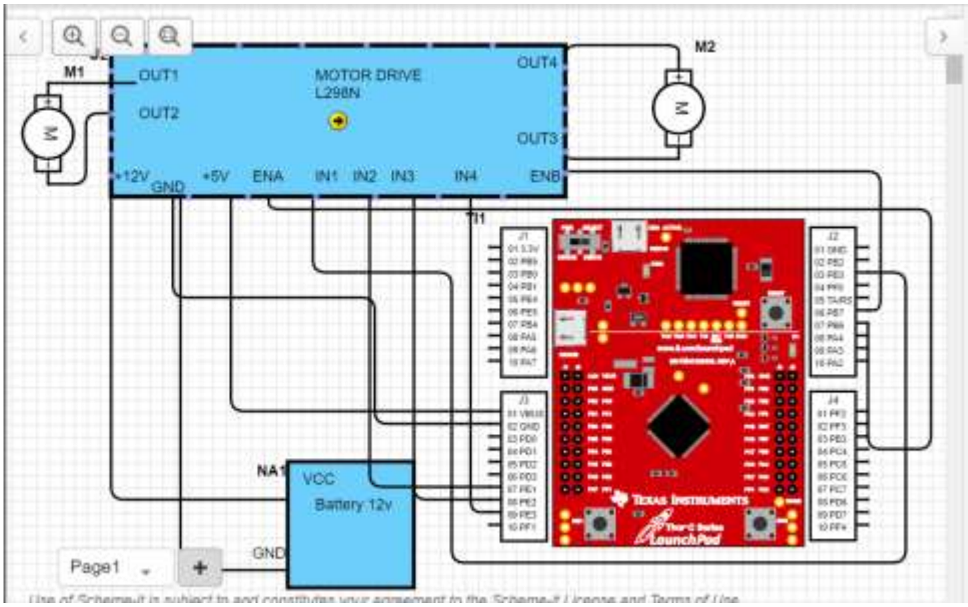
Next will be building robot car. Connect PWM outputs to the two DC motors driver PWM signals, test speed control using onboard push button SW1. Add direction control to the robot car: figure out which pins use for direction control. Modify function GPIOPortF_Handler() SW2 touch logic to change direction of the motors. Add LED indicator for speed and direction: Add GPIO initialization code to use the three onboard LEDs. Modify GPIOPortF_Handler() SW1, SW2 touch logic to take care of direction LED changes.

For this project, I am using a L298N motor driver which allows me to control the speed and direction of two DC motors. The L298N H-bridge module can be used with motors that have a voltage of between 5 and 35V DC. There is also an onboard 5V regulator, so because my supply voltage is up to 12V, I sourced 5V from the board.

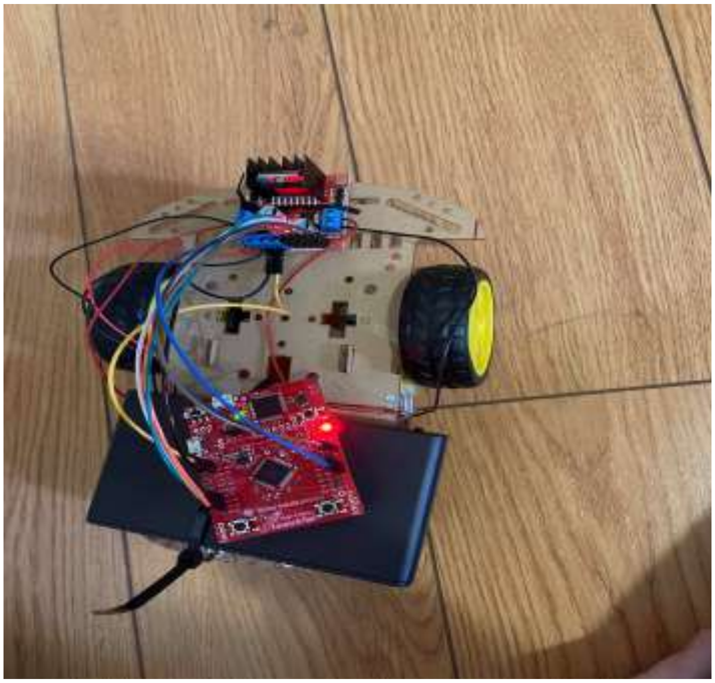
To control one or two DC motors: First, connect each motor to the A and B connections on the L298N module. The polarity of the motors should the same on both inputs. Next, connect power supply - the positive to pin 4 on the module and negative/GND to pin 5. Because my supply is up to 12V, I leave in the 12V jumper and 5V will be available from pin 6 on the module. This will be fed to LaunchPad vbus pin to power it from the motors' power supply. Then connect LaunchPad GND to pin 5 on the module to complete the circuit.

Two PWM signals connected to pin 7 and 12 respectively will be used to control the speed of the two DC motors. Four digital signal signals connected to In1 – In 4 will be used to control the direction of the two DC motors: each motor needs two digital signals, send a HIGH to IN1 and a LOW to IN2 will cause it to turn in one direction, and a LOW and HIGH will cause it to turn in the other direction.

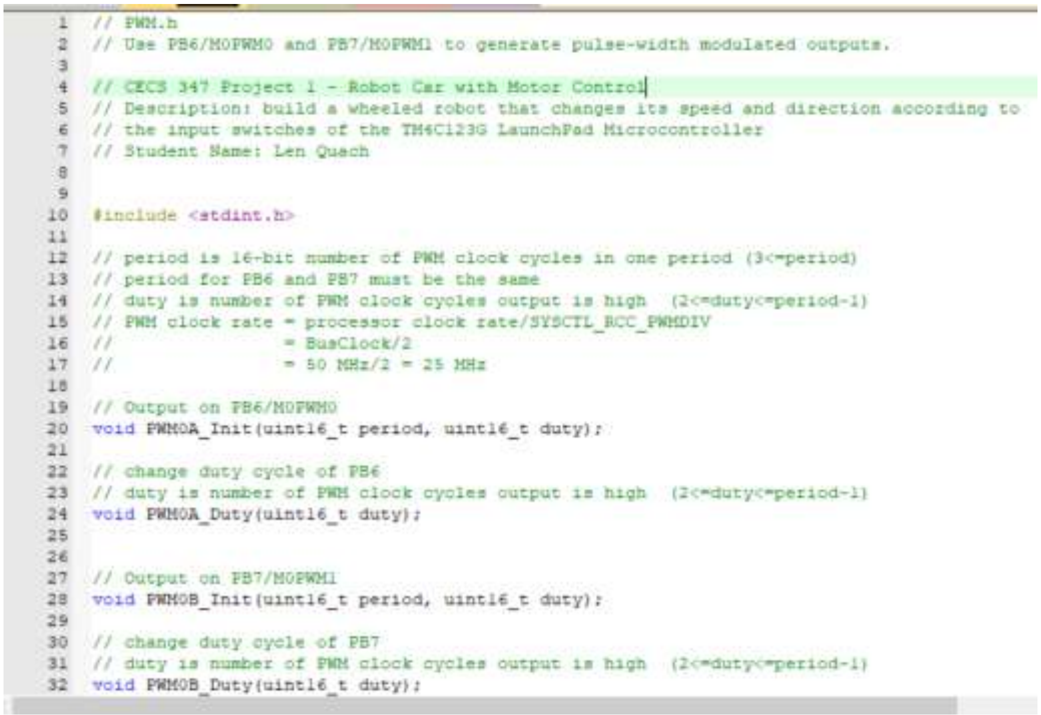
Hardware design



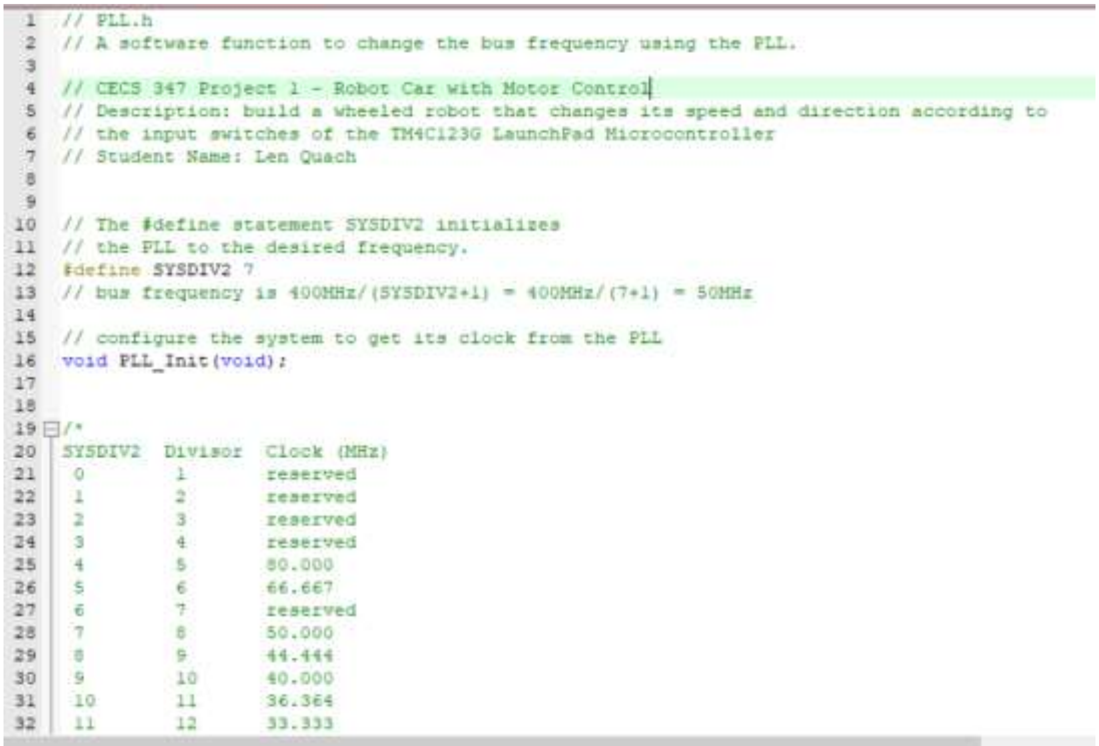
Use of Scheme-it is subject to and constitutes your agreement to the Scheme-it License and Terms of Use



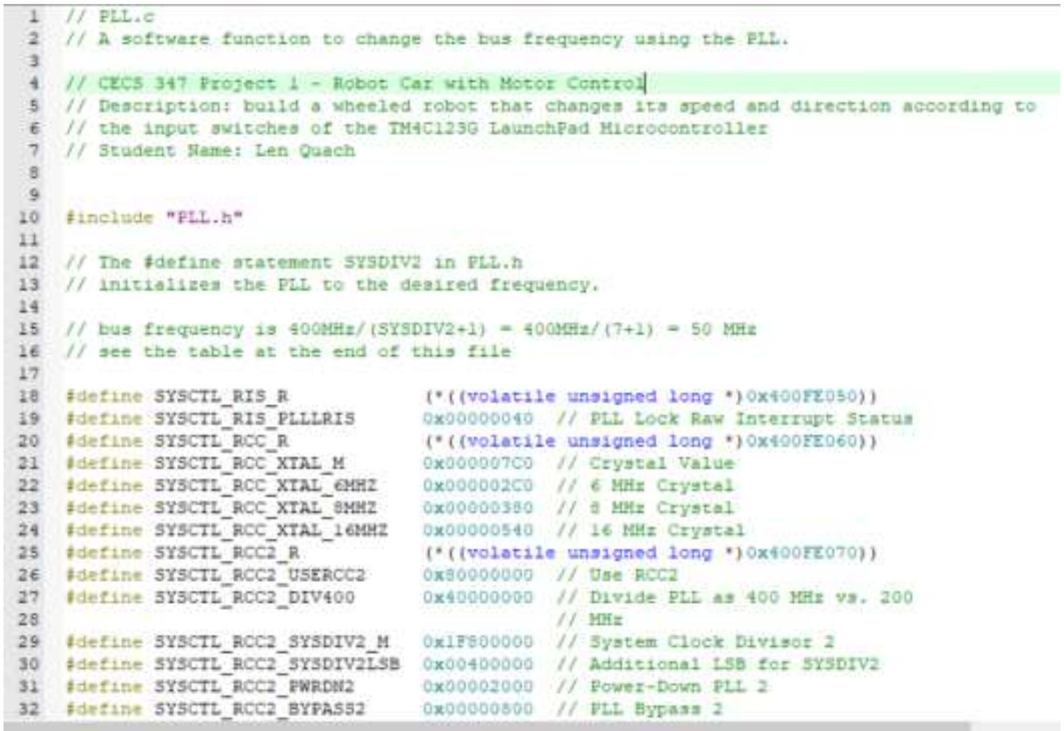
Software design



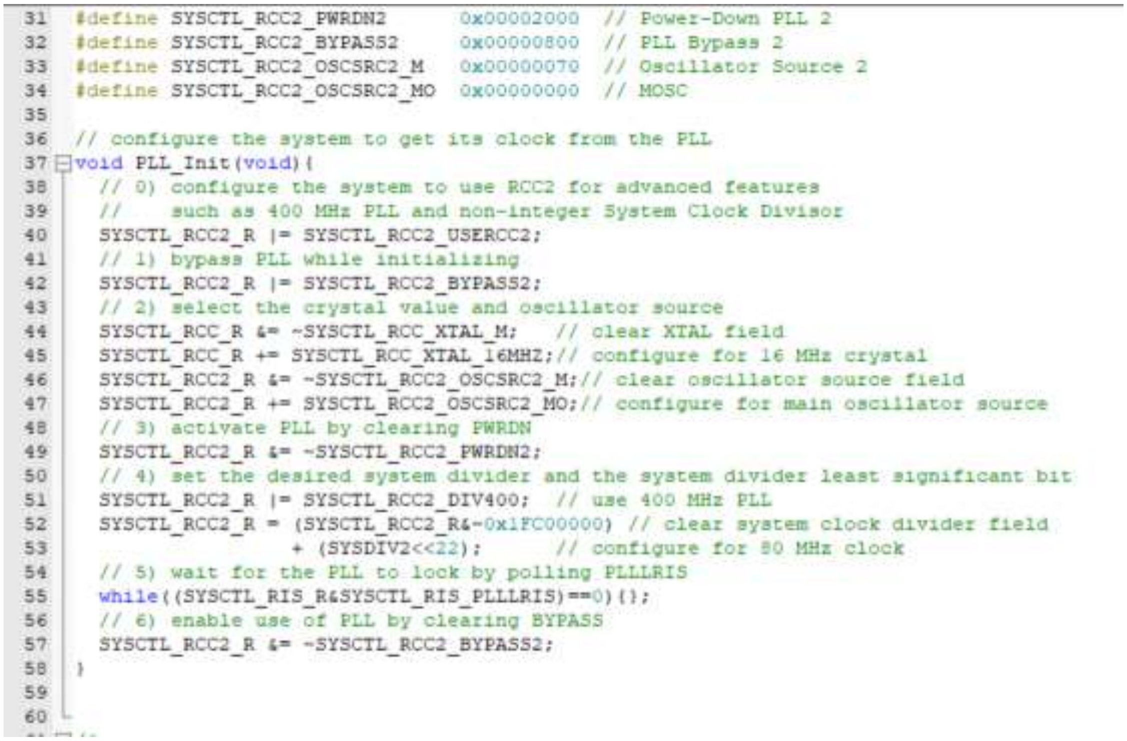
```c
// PWM.h
// Use PB6/M0PWM0 and PB7/M0PWM1 to generate pulse-width modulated outputs.

// CECS 347 Project 1 - Robot Car with Motor Control
// Description: build a wheeled robot that changes its speed and direction according to
// the input switches of the TM4C123G LaunchPad Microcontroller
// Student Name: Len Quach


#include <stdint.h>

// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                = BusClock/2
//                = 50 MHz/2 = 25 MHz

// Output on PB6/M0PWM0
void PWM0A_Init(uint16_t period, uint16_t duty);

// change duty cycle of PB6
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0A_Duty(uint16_t duty);


// Output on PB7/M0PWM1
void PWM0B_Init(uint16_t period, uint16_t duty);

// change duty cycle of PB7
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0B_Duty(uint16_t duty);
```



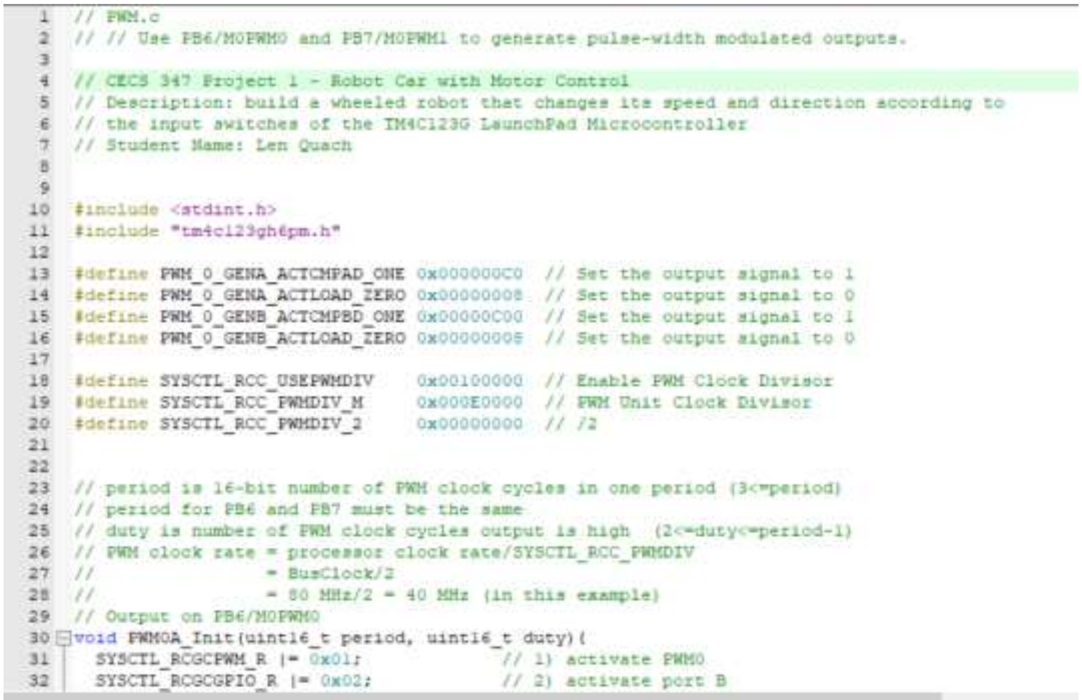
```
// PLL.h
// A software function to change the bus frequency using the PLL.

// CECS 347 Project 1 - Robot Car with Motor Control
// Description: build a wheeled robot that changes its speed and direction according to
// the input switches of the TM4C123G LaunchPad Microcontroller
// Student Name: Len Quach


// The #define statement SYSDIV2 initializes
// the PLL to the desired frequency.
#define SYSDIV2 7
// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50MHz

// configure the system to get its clock from the PLL
void PLL_Init(void);


/*
SYSDIV2  Divisor  Clock (MHz)
 0        1       reserved
 1        2       reserved
 2        3       reserved
 3        4       reserved
 4        5       80.000
 5        6       66.667
 6        7       reserved
 7        8       50.000
 8        9       44.444
 9        10      40.000
 10       11      36.364
 11       12      33.333
```



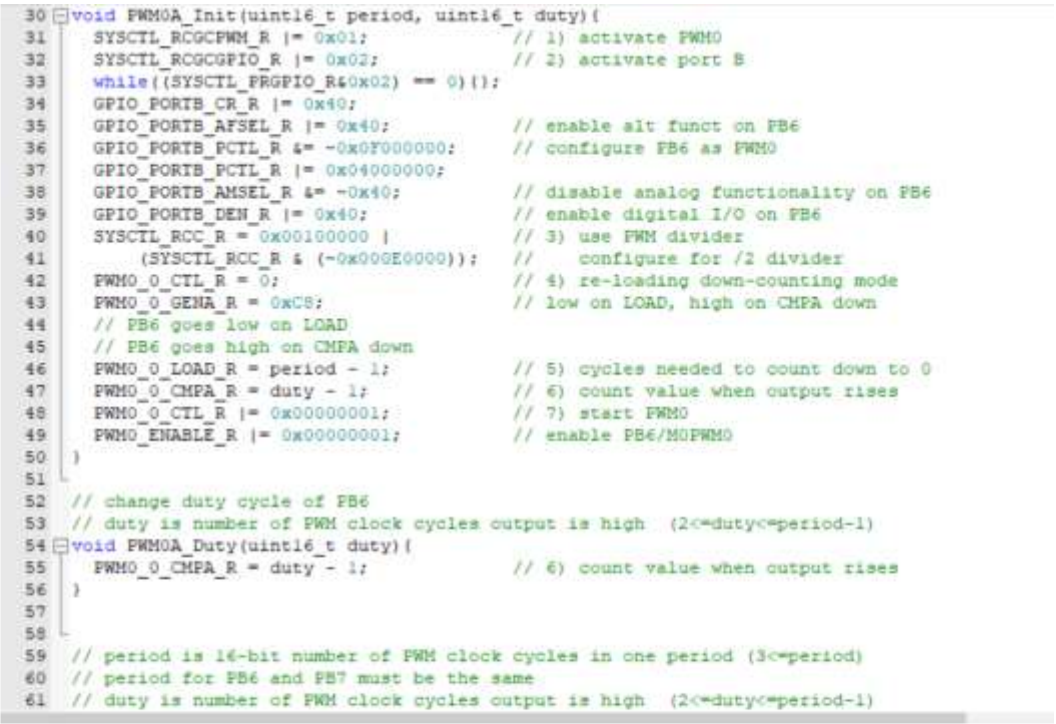
```
// PLL.c
// A software function to change the bus frequency using the PLL.

// CECS 347 Project 1 - Robot Car with Motor Control
// Description: build a wheeled robot that changes its speed and direction according to
// the input switches of the TM4C123G LaunchPad Microcontroller
// Student Name: Len Quach


#include "PLL.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.

// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_R            (*((volatile unsigned long *)0x400FE050))
#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_R            (*((volatile unsigned long *)0x400FE060))
#define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_R           (*((volatile unsigned long *)0x400FE070))
#define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
                                            // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
```



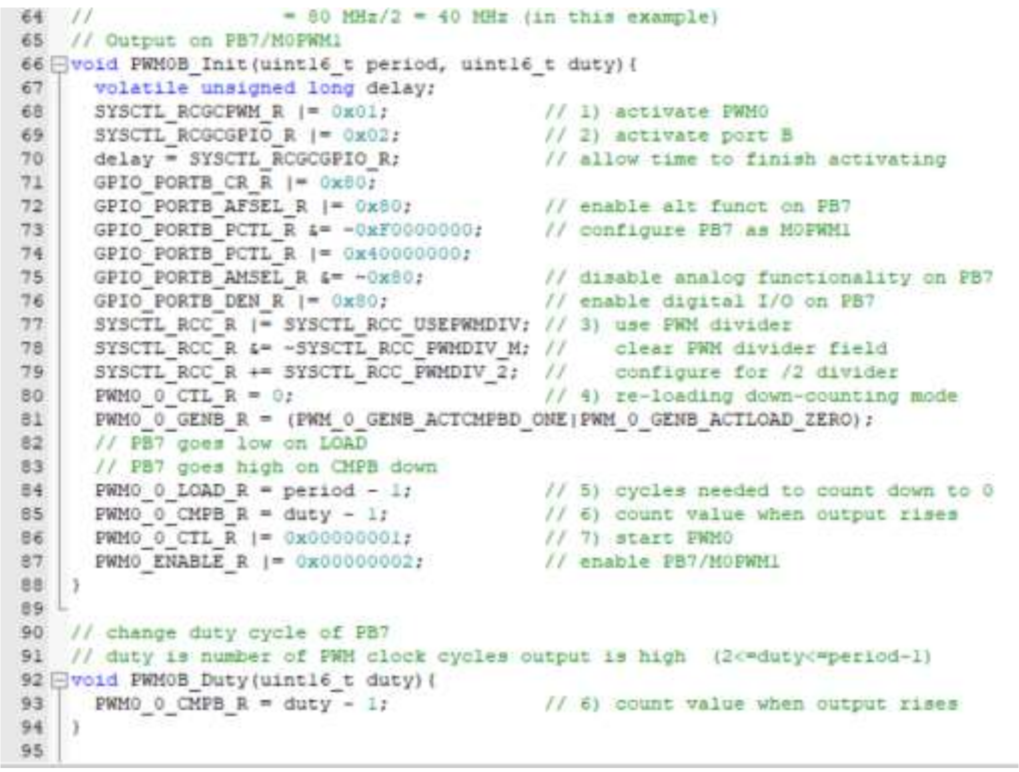
```c
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC

// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~0x1FC00000) // clear system clock divider field
                  + (SYSDIV2<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}
```



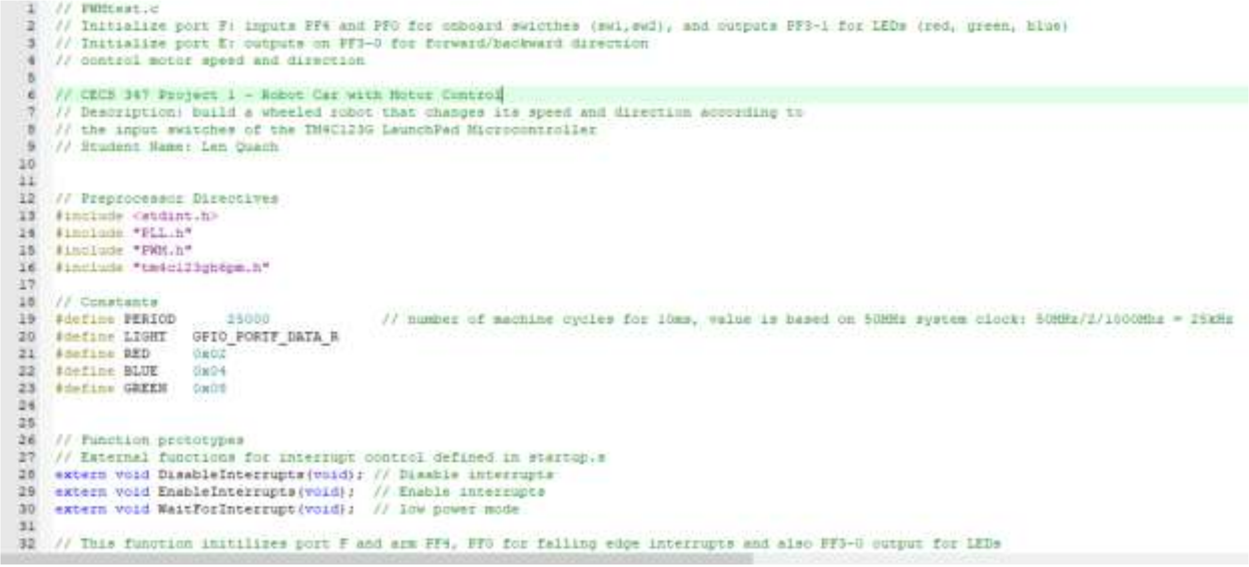
```c
// PWM.c
// // Use PB6/M0PWM0 and PB7/M0PWM1 to generate pulse-width modulated outputs.

// CECS 347 Project 1 - Robot Car with Motor Control
// Description: build a wheeled robot that changes its speed and direction according to
// the input switches of the TM4C123G LaunchPad Microcontroller
// Student Name: Len Quach


#include <stdint.h>
#include "tm4c123gh6pm.h"

#define PWM_0_GENA_ACTCMPAD_ONE 0x000000C0  // Set the output signal to 1
#define PWM_0_GENA_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0
#define PWM_0_GENB_ACTCMPBD_ONE 0x00000C00  // Set the output signal to 1
#define PWM_0_GENB_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0

#define SYSCTL_RCC_USEPWMDIV    0x00100000  // Enable PWM Clock Divisor
#define SYSCTL_RCC_PWMDIV_M     0x000E0000  // PWM Unit Clock Divisor
#define SYSCTL_RCC_PWMDIV_2     0x00000000  // /2


// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                = BusClock/2
//                = 80 MHz/2 = 40 MHz (in this example)
// Output on PB6/M0PWM0
void PWM0A_Init(uint16_t period, uint16_t duty){
  SYSCTL_RCGCPWM_R |= 0x01;             // 1) activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // 2) activate port B
```



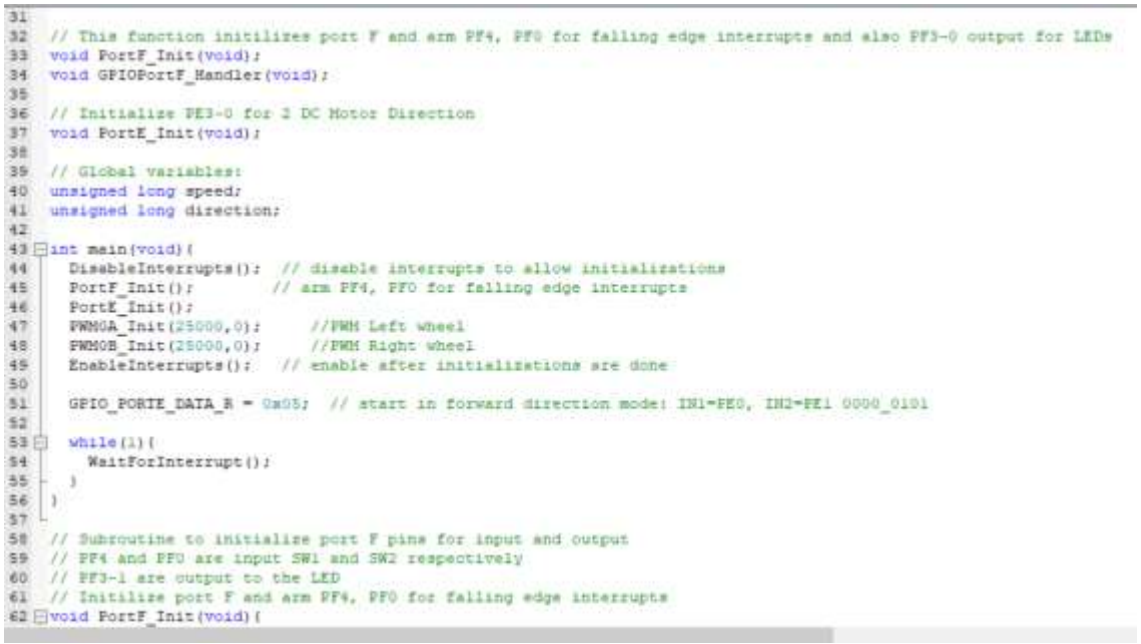
```c
void PWM0A_Init(uint16_t period, uint16_t duty){
  SYSCTL_RCGCPWM_R |= 0x01;             // 1) activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // 2) activate port B
  while((SYSCTL_PRGPIO_R&0x02) == 0){};
  GPIO_PORTB_CR_R |= 0x40;
  GPIO_PORTB_AFSEL_R |= 0x40;           // enable alt funct on PB6
  GPIO_PORTB_PCTL_R &= ~0x0F000000;     // configure PB6 as PWM0
  GPIO_PORTB_PCTL_R |= 0x04000000;
  GPIO_PORTB_AMSEL_R &= ~0x40;          // disable analog functionality on PB6
  GPIO_PORTB_DEN_R |= 0x40;             // enable digital I/O on PB6
  SYSCTL_RCC_R = 0x00100000 |           // 3) use PWM divider
      (SYSCTL_RCC_R & (~0x000E0000));   //    configure for /2 divider
  PWM0_0_CTL_R = 0;                     // 4) re-loading down-counting mode
  PWM0_0_GENA_R = 0xC8;                 // low on LOAD, high on CMPA down
  // PB6 goes low on LOAD
  // PB6 goes high on CMPA down
  PWM0_0_LOAD_R = period - 1;           // 5) cycles needed to count down to 0
  PWM0_0_CMPA_R = duty - 1;             // 6) count value when output rises
  PWM0_0_CTL_R |= 0x00000001;           // 7) start PWM0
  PWM0_ENABLE_R |= 0x00000001;          // enable PB6/M0PWM0
}

// change duty cycle of PB6
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0A_Duty(uint16_t duty){
  PWM0_0_CMPA_R = duty - 1;             // 6) count value when output rises
}


// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
```



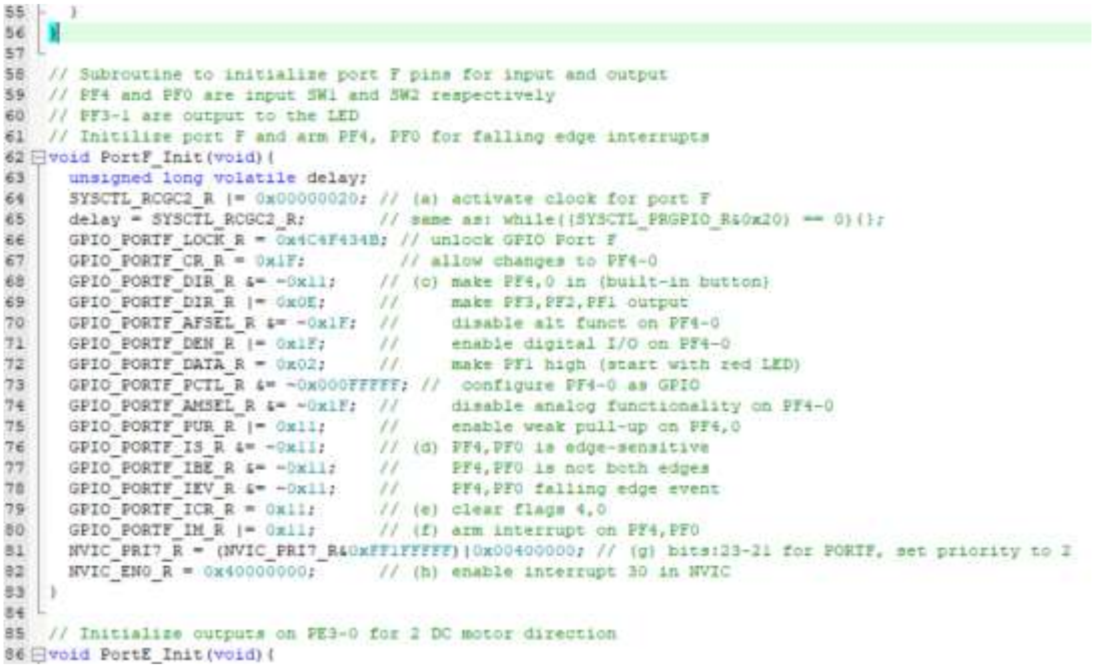
```c
//                   = 80 MHz/2 = 40 MHz (in this example)
// Output on PB7/M0PWM1
void PWM0B_Init(uint16_t period, uint16_t duty){
  volatile unsigned long delay;
  SYSCTL_RCGCPWM_R |= 0x01;             // 1) activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // 2) activate port B
  delay = SYSCTL_RCGCGPIO_R;            // allow time to finish activating
  GPIO_PORTB_CR_R |= 0x80;
  GPIO_PORTB_AFSEL_R |= 0x80;           // enable alt funct on PB7
  GPIO_PORTB_PCTL_R &= ~0xF0000000;     // configure PB7 as M0PWM1
  GPIO_PORTB_PCTL_R |= 0x40000000;
  GPIO_PORTB_AMSEL_R &= ~0x80;          // disable analog functionality on PB7
  GPIO_PORTB_DEN_R |= 0x80;             // enable digital I/O on PB7
  SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // 3) use PWM divider
  SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; //    clear PWM divider field
  SYSCTL_RCC_R += SYSCTL_RCC_PWMDIV_2;  //    configure for /2 divider
  PWM0_0_CTL_R = 0;                     // 4) re-loading down-counting mode
  PWM0_0_GENB_R = (PWM_0_GENB_ACTCMPBD_ONE|PWM_0_GENB_ACTLOAD_ZERO);
  // PB7 goes low on LOAD
  // PB7 goes high on CMPB down
  PWM0_0_LOAD_R = period - 1;           // 5) cycles needed to count down to 0
  PWM0_0_CMPB_R = duty - 1;             // 6) count value when output rises
  PWM0_0_CTL_R |= 0x00000001;           // 7) start PWM0
  PWM0_ENABLE_R |= 0x00000002;          // enable PB7/M0PWM1
}

// change duty cycle of PB7
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0B_Duty(uint16_t duty){
  PWM0_0_CMPB_R = duty - 1;             // 6) count value when output rises
}

```



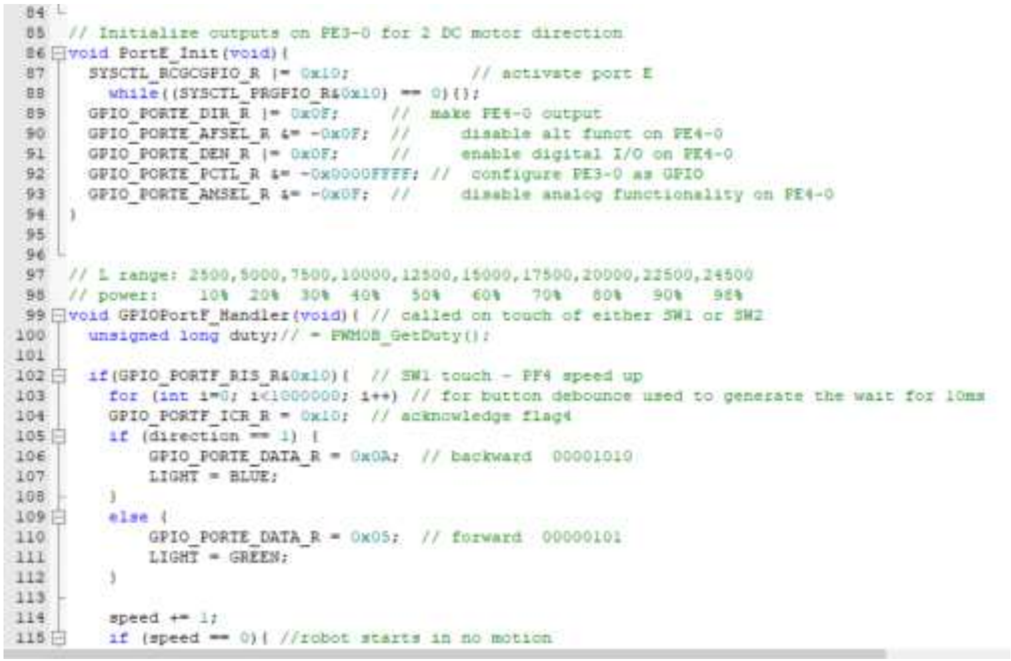
```c
// PWMtest.c
// Initialize port F: inputs PF4 and PF0 for onboard swicthes (sw1,sw2), and outputs PF3-1 for LEDs (red, green, blue)
// Initialize port E: outputs on PF3-0 for forward/backward direction
// control motor speed and direction

// CECS 347 Project 1 - Robot Car with Motor Control
// Description: build a wheeled robot that changes its speed and direction according to
// the input switches of the TM4C123G LaunchPad Microcontroller
// Student Name: Len Quach


// Preprocessor Directives
#include <stdint.h>
#include "PLL.h"
#include "PWM.h"
#include "tm4c123gh6pm.h"

// Constants
#define PERIOD      25000           // number of machine cycles for 10ms, value is based on 50MHz system clock: 50MHz/2/1000Mhz = 25kHz
#define LIGHT   GPIO_PORTF_DATA_R
#define RED     0x02
#define BLUE    0x04
#define GREEN   0x08


// Function prototypes
// External functions for interrupt control defined in startup.s
extern void DisableInterrupts(void); // Disable interrupts
extern void EnableInterrupts(void);  // Enable interrupts
extern void WaitForInterrupt(void);  // low power mode

// This function initilizes port F and arm PF4, PF0 for falling edge interrupts and also PF3-0 output for LEDs
```



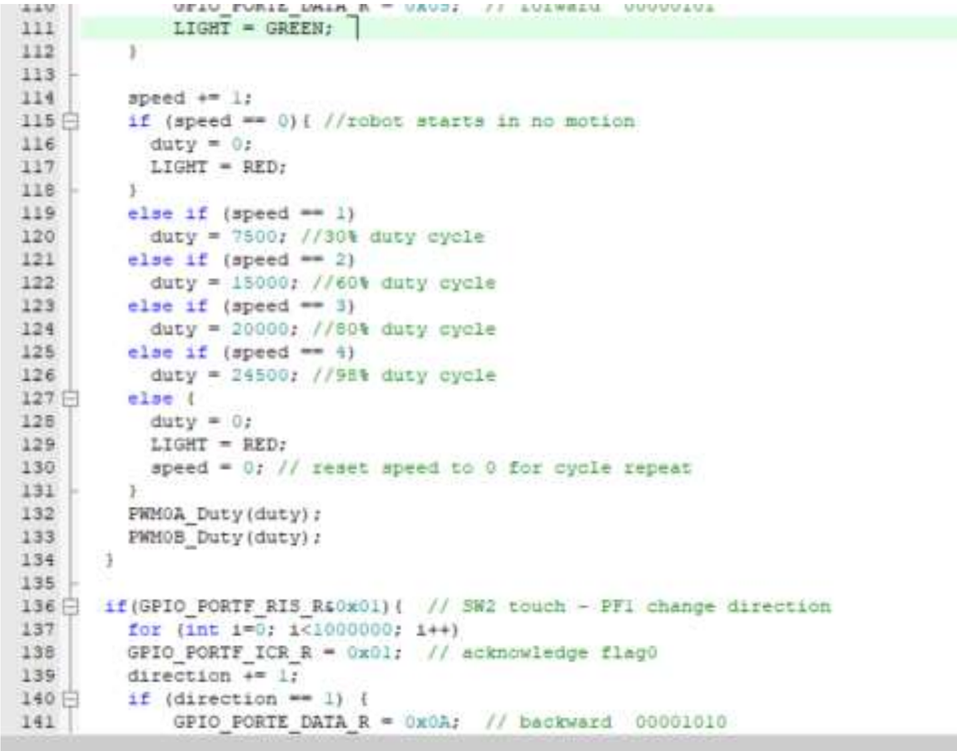
```c

// This function initilizes port F and arm PF4, PF0 for falling edge interrupts and also PF3-0 output for LEDs
void PortF_Init(void);
void GPIOPortF_Handler(void);

// Initialize PE3-0 for 2 DC Motor Direction
void PortE_Init(void);

// Global variables:
unsigned long speed;
unsigned long direction;

int main(void){
  DisableInterrupts();  // disable interrupts to allow initializations
  PortF_Init();         // arm PF4, PF0 for falling edge interrupts
  PortE_Init();
  PWM0A_Init(25000,0);      //PWM Left wheel
  PWM0B_Init(25000,0);      //PWM Right wheel
  EnableInterrupts();   // enable after initializations are done

  GPIO_PORTE_DATA_R = 0x05;  // start in forward direction mode: IN1=PE0, IN2=PE1 0000_0101

  while(1){
    WaitForInterrupt();
  }
}

// Subroutine to initialize port F pins for input and output
// PF4 and PF0 are input SW1 and SW2 respectively
// PF3-1 are output to the LED
// Initilize port F and arm PF4, PF0 for falling edge interrupts
void PortF_Init(void){
```



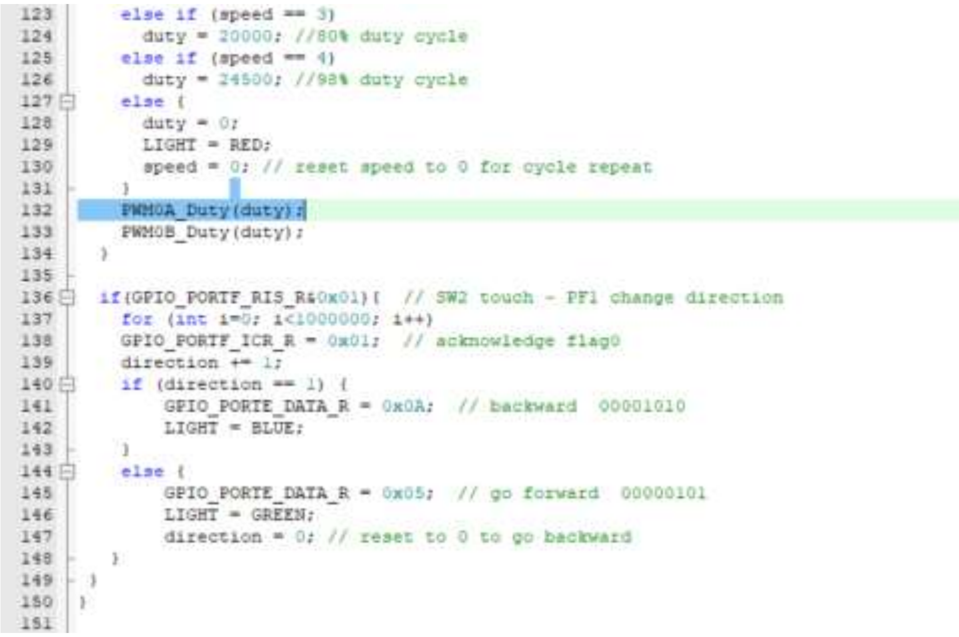
```
  }
}

// Subroutine to initialize port F pins for input and output
// PF4 and PF0 are input SW1 and SW2 respectively
// PF3-1 are output to the LED
// Initilize port F and arm PF4, PF0 for falling edge interrupts
void PortF_Init(void){
  unsigned long volatile delay;
  SYSCTL_RCGC2_R |= 0x00000020; // (a) activate clock for port F
  delay = SYSCTL_RCGC2_R;       // same as: while((SYSCTL_PRGPIO_R&0x20) == 0){};
  GPIO_PORTF_LOCK_R = 0x4C4F434B; // unlock GPIO Port F
  GPIO_PORTF_CR_R = 0x1F;         // allow changes to PF4-0
  GPIO_PORTF_DIR_R &= ~0x11;    // (c) make PF4,0 in (built-in button)
  GPIO_PORTF_DIR_R |= 0x0E;     //     make PF3,PF2,PF1 output
  GPIO_PORTF_AFSEL_R &= ~0x1F;  //     disable alt funct on PF4-0
  GPIO_PORTF_DEN_R |= 0x1F;     //     enable digital I/O on PF4-0
  GPIO_PORTF_DATA_R = 0x02;     //     make PF1 high (start with red LED)
  GPIO_PORTF_PCTL_R &= ~0x000FFFFF; //  configure PF4-0 as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x1F;  //     disable analog functionality on PF4-0
  GPIO_PORTF_PUR_R |= 0x11;     //     enable weak pull-up on PF4,0
  GPIO_PORTF_IS_R &= ~0x11;     // (d) PF4,PF0 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x11;    //     PF4,PF0 is not both edges
  GPIO_PORTF_IEV_R &= ~0x11;    //     PF4,PF0 falling edge event
  GPIO_PORTF_ICR_R = 0x11;      // (e) clear flags 4,0
  GPIO_PORTF_IM_R |= 0x11;      // (f) arm interrupt on PF4,PF0
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00400000; // (g) bits:23-21 for PORTF, set priority to 2
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
}

// Initialize outputs on PE3-0 for 2 DC motor direction
void PortE_Init(void){
```



```

// Initialize outputs on PE3-0 for 2 DC motor direction
void PortE_Init(void){
  SYSCTL_RCGCGPIO_R |= 0x10;             // activate port E
    while((SYSCTL_PRGPIO_R&0x10) == 0){};
  GPIO_PORTE_DIR_R |= 0x0F;     //  make PE4-0 output
  GPIO_PORTE_AFSEL_R &= ~0x0F;  //     disable alt funct on PE4-0
  GPIO_PORTE_DEN_R |= 0x0F;     //     enable digital I/O on PE4-0
  GPIO_PORTE_PCTL_R &= ~0x0000FFFF; //  configure PE3-0 as GPIO
  GPIO_PORTE_AMSEL_R &= ~0x0F;  //     disable analog functionality on PE4-0
}


// L range: 2500,5000,7500,10000,12500,15000,17500,20000,22500,24500
// power:    10%  20%  30%  40%   50%   60%   70%   80%   90%   98%
void GPIOPortF_Handler(void){ // called on touch of either SW1 or SW2
  unsigned long duty;// = PWM0B_GetDuty();

  if(GPIO_PORTF_RIS_R&0x10){  // SW1 touch - PF4 speed up
    for (int i=0; i<1000000; i++) // for button debounce used to generate the wait for 10ms
    GPIO_PORTF_ICR_R = 0x10;  // acknowledge flag4
    if (direction == 1) {
        GPIO_PORTE_DATA_R = 0x0A;  // backward  00001010
        LIGHT = BLUE;
    }
    else {
        GPIO_PORTE_DATA_R = 0x05;  // forward  00000101
        LIGHT = GREEN;
    }

    speed += 1;
    if (speed == 0){ //robot starts in no motion
```



```
        GPIO_PORTE_DATA_R = 0x05;  // forward  00000101
        LIGHT = GREEN;
    }

    speed += 1;
    if (speed == 0){ //robot starts in no motion
      duty = 0;
      LIGHT = RED;
    }
    else if (speed == 1)
      duty = 7500; //30% duty cycle
    else if (speed == 2)
      duty = 15000; //60% duty cycle
    else if (speed == 3)
      duty = 20000; //80% duty cycle
    else if (speed == 4)
      duty = 24500; //98% duty cycle
    else {
      duty = 0;
      LIGHT = RED;
      speed = 0; // reset speed to 0 for cycle repeat
    }
    PWM0A_Duty(duty);
    PWM0B_Duty(duty);
  }

  if(GPIO_PORTF_RIS_R&0x01){  // SW2 touch - PF1 change direction
    for (int i=0; i<1000000; i++)
    GPIO_PORTF_ICR_R = 0x01;  // acknowledge flag0
    direction += 1;
    if (direction == 1) {
        GPIO_PORTE_DATA_R = 0x0A;  // backward  00001010
```



```
    else if (speed == 3)
      duty = 20000; //80% duty cycle
    else if (speed == 4)
      duty = 24500; //98% duty cycle
    else {
      duty = 0;
      LIGHT = RED;
      speed = 0; // reset speed to 0 for cycle repeat
    }
    PWM0A_Duty(duty);
    PWM0B_Duty(duty);
  }

  if(GPIO_PORTF_RIS_R&0x01){  // SW2 touch - PF1 change direction
    for (int i=0; i<1000000; i++)
    GPIO_PORTF_ICR_R = 0x01;  // acknowledge flag0
    direction += 1;
    if (direction == 1) {
        GPIO_PORTE_DATA_R = 0x0A;  // backward  00001010
        LIGHT = BLUE;
    }
    else {
        GPIO_PORTE_DATA_R = 0x05;  // go forward  00000101
        LIGHT = GREEN;
        direction = 0; // reset to 0 to go backward
   }
 }
}
```

Conclusion

The project of robot car with motor control helps me practice how to use hardware PWM and PLL hardware components and review GPIO and interrupts topics that was covered in CECS 346. One of the problems I had in this project was the program could not download the code to the board due to board frying. But it was good to know that I should have disconnected the wires connected to vbus while downloading the code to the board.