```c
/*
*Name: Kenzie Moore
* Program: Electrical Engineering Technology
* Year: 3rd year
* Class: Microcontroller Systems
* Section: CPET 253
* Exercise: Lab 5 Prelab
* Date : 2/12/2022
*/


// Motor.c
// Runs on MSP432
// Provide mid-level functions that initialize ports and
// set motor speeds to move the robot. Lab 13 solution
// Daniel Valvano
// July 11, 2019

/* This example accompanies the book
   "Embedded Systems: Introduction to Robotics,
   Jonathan W. Valvano, ISBN: 9781074544300, copyright (c) 2019
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/

Simplified BSD License (FreeBSD License)
Copyright (c) 2019, Jonathan Valvano, All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are
those of the authors and should not be interpreted as representing official
policies, either expressed or implied, of the FreeBSD Project.
*/

// Left motor direction connected to P5.4 (J3.29)
// Left motor PWM connected to P2.7/TA0CCP4 (J4.40)
// Left motor enable connected to P3.7 (J4.31)
// Right motor direction connected to P5.5 (J3.30)
// Right motor PWM connected to P2.6/TA0CCP3 (J4.39)
// Right motor enable connected to P3.6 (J2.11)

#include <stdint.h>
#include "msp.h"
#include "../inc/CortexM.h"
#include "../inc/PWM.h"

// *******Lab 4 Solution*******

// ------------Motor_Init------------
// Initialize GPIO pins for output, which will be
// used to control the direction of the motors and
// to enable or disable the drivers.
// The motors are initially stopped, the drivers
// are initially powered down, and the PWM speed
// control is uninitialized.
// Input: none
// Output: none
#define RIGHT_MOT_DIR      0x20     //p5.5
#define RIGHT_MOT_SLEEP    0x40     //p3.6
#define RIGHT_MOT_PWM      0x40     //p2.6
#define LEFT_MOT_DIR       0x10     //p5.4
#define LEFT_MOT_SLEEP     0x80     //p3.7
#define LEFT_MOT_PWM       0x80     //p2.7

void Motor_Init(void){
    //set direction pins as outputs
    P5DIR |= RIGHT_MOT_DIR | LEFT_MOT_DIR;
    //set PWM pins as outputs
    P3DIR |= RIGHT_MOT_PWM | LEFT_MOT_PWM;
    //set sleep pins as outputs
    P2DIR |= RIGHT_MOT_SLEEP | LEFT_MOT_SLEEP;
    //put motors to sleep
    P3OUT &= ~RIGHT_MOT_SLEEP & ~LEFT_MOT_SLEEP;

    return;
}

// ------------Motor_Stop------------
// Stop the motors, power down the drivers, and
// set the PWM speed control to 0% duty cycle.
// Input: none
// Output: none
void Motor_Stop(void){
    P2OUT &= ~RIGHT_MOT_PWM & ~LEFT_MOT_PWM;       //stop motors
    P3OUT &= ~RIGHT_MOT_SLEEP & ~LEFT_MOT_SLEEP;   //put motors to sleep
    return;
}
///////////////////////////////////////////////////////////////////////////////////
void TimerInit(void)
{
//First initialize TimerA0 for PWM
    P2DIR |= 0x40; // MAKE 2.6 OUTPUT
    P2SEL1 &= ~0x40;
    P2SEL0 |= 0x40;

    P2DIR |= 0x80; // MAKE 2.7 OUTPUT
    P2SEL1 &= ~0x80;
    P2SEL0 |= 0x80;

    TA0CCR0 = 59999;//Since the motors are connected to P2.6 and P2.7, use TimerA0, compare blocks 3 & 4
    TA0CCR3 = 14999;
    TA0CCR4 = 14999;
    TA0CTL |= 0x0010; //SET TIMER FOR UP MODE - this starts it

    TA0CTL &= ~0x0030;  //stop the timer
    TA0CTL |= 0x0200;  TA0CTL &= ~0x0100;//choose SMCLK for the clock source

    TA0CTL |= 0x0040; TA0CTL &= ~0x0080;//choose clock divider of 2
    TA0CCTL3 |= 0x00E0; //Outmode 7: reset/set
    TA0CCTL4 |= 0x00E0;    //Outmode 7: reset/set

}
////////////////////////////////////////////////////////////////////
```

```c
void Delay10u(void){

        //Now initialize TimerAx for the delay function
        //10us delay

        TA2CTL &= ~0x0030;  //stop the timer
        TA2CTL |= 0x0200; TA2CTL &= ~0x0100;    //choose SMCLK for the clock source
        TA2CTL |= 0x0040; TA2CTL &= ~0x0080;    //choose clock divider of 2 : ID = 01
        TA2CCR0 = 59;                     //
        TA2R = 0;                            //clear timer
        TA2CTL |= 0x0010;
        while(!(TA2CCTL0 & 0x0001)){}
        TA2CCTL0 &= ~0x0001; //clear the flag
        TA2CTL &= ~0x0030;  //stop the timer
return;
}
//////////////////////////////////////////////////////////////////////
void Delay100(void){

        //Now initialize TimerAx for the delay function


        TA3CTL &= ~0x0030;  //stop the timer
        TA3CTL |= 0x0200; TA2CTL &= ~0x0100;    //choose SMCLK for the clock source
        TA3CTL |= 0x0080; TA2CTL &= ~0x0040;    //choose clock divider of 4 : ID = 10
        TA3EX0 |= 0x0004; TA2EX0 &= ~0x0003; //choose second clock divider in TAxEX0 of 5, total divide is 20
        TA3CCR0 = 59999;                     //
        TA3R = 0;                            //clear timer
        TA3CTL |= 0x0010;
        while(!(TA3CCTL0 & 0x0001)){}
        TA3CCTL0 &= ~0x0001; //clear the flag
        TA3CTL &= ~0x0030;  //stop the timer
return;
}
/////////////////////////////////////////////////////////////////////////////////////////////
// ------------Motor_Forward------------
// Drive the robot forward by running left and
// right wheels forward with the given duty
// cycles.
// Input: duty1  duty cycle of left wheel (0 to 14,998)
//        duty2 duty cycle of right wheel (0 to 14,998)
// Output: none
// Assumes: Motor_Init() has been called
void Motor_Forward(uint16_t leftDuty, uint16_t rightDuty){
    // Run TimerA0 in PWM mode with provided duty cycle
    // Set motor controls for forward

    // turn on PWM and set duty cycle
    // fixed period of 10ms
    TA0CTL |= 0x0010; // Control bits 5 and 4 are mode control 00 to stop, 01 for up counting
                    // bits 7 and 6 are clock divider 01 = /2
                    // bits 9 and 8 choose clock 10 = SMCLK

    TA0R = 0;       // Counter, start at zero once turned on
    TA0CCR3 = leftDuty; // Capture/Compare 3 COMPARE MODE : holds value for comparison to timer TA0R
    TA0CCR4 = rightDuty; // Capture/Compare 4 COMPARE MODE : holds value for comparison to timer TA0R

    //left motor - START
    P5OUT &= ~0b00010000; //DIRL on P5.4 (PH)
    P2OUT |= 0b10000000; //PWML on P2.7 (EN)
    P3OUT |= 0b10000000; //nSLPL on P3.7(nSLEEP)
    //right motor - START
    P5OUT &= ~0b00100000; //DIRR on P5.5 (PH)
    P2OUT |= 0b01000000; //PWMR on P2.6 (EN)
    P3OUT |= 0b01000000; //nSLPR on P3.6(nSLEEP)

return;
}

// ------------Motor_Right------------
// Turn the robot to the right by running the
// left wheel forward and the right wheel
// backward with the given duty cycles.
// Input: duty1  duty cycle of left wheel (0 to 14,998)
//        duty2 duty cycle of right wheel (0 to 14,998)
// Output: none
// Assumes: Motor_Init() has been called
void Motor_Right(uint16_t leftDuty, uint16_t rightDuty){
    // Run TimerA0 in PWM mode with provided duty cycle
    // Set motor controls for forward

    // turn on PWM and set duty cycle
    // fixed period of 10ms
    TA0CTL |= 0x0010; // Control bits 5 and 4 are mode control 00 to stop, 01 for up counting
                    // bits 7 and 6 are clock divider 01 = /2
                    // bits 9 and 8 choose clock 10 = SMCLK

    TA0R = 0;       // Counter, start at zero once turned on
    TA0CCR3 = leftDuty; // Capture/Compare 3 COMPARE MODE : holds value for comparison to timer TA0R
    TA0CCR4 = rightDuty; // Capture/Compare 4 COMPARE MODE : holds value for comparison to timer TA0R

    //left motor - START
    P5OUT &= ~0b00010000; //DIRL on P5.4 (PH)
    P2OUT |= 0b10000000; //PWML on P2.7 (EN)
    P3OUT |= 0b10000000; //nSLPL on P3.7(nSLEEP)

    //right motor - START
    P5OUT |= 0b00100000; //DIRR on P5.5 (PH)
    P2OUT |= 0b01000000; //PWMR on P2.6 (EN)
    P3OUT |= 0b01000000; //nSLPR on P3.6(nSLEEP)

return;
}

// ------------Motor_Left------------
// Turn the robot to the left by running the
// left wheel backward and the right wheel
// forward with the given duty cycles.
// Input: duty1  duty cycle of left wheel (0 to 14,998)
//        duty2 duty cycle of right wheel (0 to 14,998)
// Output: none
// Assumes: Motor_Init() has been called
void Motor_Left(uint16_t leftDuty, uint16_t rightDuty){
    // Run TimerA0 in PWM mode with provided duty cycle
    // Set motor controls for forward

    // turn on PWM and set duty cycle
    // fixed period of 10ms
    TA0CTL |= 0x0010; // Control bits 5 and 4 are mode control 00 to stop, 01 for up counting
                    // bits 7 and 6 are clock divider 01 = /2
                    // bits 9 and 8 choose clock 10 = SMCLK

    TA0R = 0;       // Counter, start at zero once turned on
    TA0CCR3 = leftDuty; // Capture/Compare 3 COMPARE MODE : holds value for comparison to timer TA0R
    TA0CCR4 = rightDuty; // Capture/Compare 4 COMPARE MODE : holds value for comparison to timer TA0R

    //left motor - START
    P5OUT |= 0b00010000; //DIRL on P5.4 (PH)
    P2OUT |= 0b10000000; //PWML on P2.7 (EN)
    P3OUT |= 0b10000000; //nSLPL on P3.7(nSLEEP)

    //right motor - START
    P5OUT &= ~0b00100000; //DIRR on P5.5 (PH)
    P2OUT |= 0b01000000; //PWMR on P2.6 (EN)
    P3OUT |= 0b01000000; //nSLPR on P3.6(nSLEEP)

return;

}

// ------------Motor_Backward------------
// Drive the robot backward by running left and
// right wheels backward with the given duty
// cycles.
// Input: duty1  duty cycle of left wheel (0 to 14,998)
//        duty2 duty cycle of right wheel (0 to 14,998)
// Output: none
// Assumes: Motor_Init() has been called
void Motor_Backward(uint16_t leftDuty, uint16_t rightDuty){
    // Run TimerA0 in PWM mode with provided duty cycle
      // Set motor controls for forward

      // turn on PWM and set duty cycle
      // fixed period of 10ms
      TA0CTL |= 0x0010; // Control bits 5 and 4 are mode control 00 to stop, 01 for up counting
                      // bits 7 and 6 are clock divider 01 = /2
                      // bits 9 and 8 choose clock 10 = SMCLK

      TA0R = 0;       // Counter, start at zero once turned on
      TA0CCR3 = leftDuty; // Capture/Compare 3 COMPARE MODE : holds value for comparison to timer TA0R
      TA0CCR4 = rightDuty; // Capture/Compare 4 COMPARE MODE : holds value for comparison to timer TA0R

      //left motor - START
      P5OUT |= 0b00010000; //DIRL on P5.4 (PH)
      P2OUT |= 0b10000000; //PWML on P2.7 (EN)
      P3OUT |= 0b10000000; //nSLPL on P3.7(nSLEEP)

      //right motor - START
      P5OUT |= 0b00100000; //DIRR on P5.5 (PH)
      P2OUT |= 0b01000000; //PWMR on P2.6 (EN)
      P3OUT |= 0b01000000; //nSLPR on P3.6(nSLEEP)

  return;
}
```