```
//-------------------------Project Details----------------------------------------
// Project: Initial program for different modules QEI, PWM and Interrupt for motor control
//
// Author:R Yogeshwar Rao
//
// Date: 1 Oct 2024

//Header Files
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/pwm.h"
#include "inc/hw_ints.h"
#include "driverlib/qei.h"
#include <time.h>
#include <math.h>
#include "driverlib/adc.h"
#define PWM_FREQUENCY 10000 ///PWM frequency in terms of Hz
```

```
int32_t xMax=14808;
int32_t xDesired=14808;
int32_t xOld=0;
int32_t Kp=100;
int32_t Kd=1;



//Function Prototypes
void Timer_Init(void);
void Hardware_Init(void);
void PWM_Init(void);
void QEI_Init(void);

//Encoder Variables and functions
signed int read_encoder(void);
int32_t En_Read;
int32_t position;
int32_t pos_max;

int32_t velocity;
int32_t rpm;

int ui32Load;

int ppr = 500;
signed int direction=0;
```

```
//Global Variables


int32_t widthAdjust;
int i = 0;
int64_t A;
int D = 1;
int32_t count=0;
int32_t newDuty;
float errorOld=0;
float errorSum=0;

uint8_t ui8PinData=2;




/**
  * main.c
  */
int main(void)
{
        Hardware_Init(); // hardware Configure
        PWM_Init();          // PWM configuration
        QEI_Init();          // QEI configuration
        Timer_Init();
        while(1)
        {


        }
```

```
}


//Configure Hardware
void Hardware_Init(void)
{
    //Configure System Clock and PWM Clock
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);


    //Enable the peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //GPIO F enable for PWM
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // enable Pin for motor direction
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); //PWM1
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); // making PD1, PD2, PD3 as output for motor direction


    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);// Enable PORT D as QEI Module 0 is peripheral on Port D
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI0); // Enable QEI Module 0


}


//Enable and configure Timer Interrupts
void Timer_Init()
{
    uint32_t ui32Period;
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    ui32Period = (SysCtlClockGet()*0.001);    // Setting the loop time to 0.1 s
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
```

```
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_A);
}

//Initialize PWM Module
void PWM_Init(void)
{
    GPIOPinTypePWM(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); //Port F as PWM

    GPIOPinConfigure(GPIO_PF2_M1PWM6);    //Blue //GPIO PF2 as M1PWM6 PWMModule 1 with PWM Generator 6

    PWMGenConfigure(PWM1_BASE,PWM_GEN_3,PWM_GEN_MODE_DOWN); //PWM Generator Configuration
    SysCtlPWMClockSet(SYSCTL_PWMDIV_64); //Division factor 64

    ui32Load = (SysCtlPWMClockGet() / PWM_FREQUENCY) - 1 ; // calculating period in pwm clock ticks

    PWMGenPeriodSet(PWM1_BASE,PWM_GEN_3,ui32Load); //written in PWM clock ticks

    PWMOutputState(PWM1_BASE,PWM_OUT_6_BIT,true);

    PWMGenEnable(PWM1_BASE,PWM_GEN_3); //Enable PWM Generator 3

    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1|GPIO_PIN_2, 2); // Direction enable for motor
```

```
}

//Initialize QEI Module
void QEI_Init(void)
{
    //Unlock GPIOD7 - Without this step it doesn't work
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= 0x80;
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;

    //Configure the QEI Pins
    GPIOPinTypeQEI(GPIO_PORTD_BASE, GPIO_PIN_6 | GPIO_PIN_7);
    GPIOPinConfigure(GPIO_PD6_PHA0);
    GPIOPinConfigure(GPIO_PD7_PHB0);

    //Disable peripheral and INT before configuration
    QEIDisable(QEI0_BASE);
    QEIIntDisable(QEI0_BASE,QEI_INTERROR | QEI_INTDIR | QEI_INTTIMER|QEI_INTINDEX);

    //Configure QEI
    pos_max = 3200000;
    QEIConfigure(QEI0_BASE, (QEI_CONFIG_CAPTURE_A_B | QEI_CONFIG_NO_RESET
|QEI_CONFIG_QUADRATURE | QEI_CONFIG_NO_SWAP), pos_max);
    //QEI_CONFIG_CAPTURE_A_B: Capture mode, here capture edges on both channels
    //QEI_CONFIG_RESET_IDX: whether to reset position integrator on index pulse detection or not
    //QEI_CONFIG_QUADRATURE: Specify QEI Mode, whether Quadrature or Clock/Direction
    //QEI_CONFIG_NO_SWAP: whether to swap channel A and B before processing or not
    //32000: maximum value of the position integrator. This is used to reset position integrator
```

```
    //Enables the QEI module
    QEIEnable(QEI0_BASE);



    //Configure the velocity capture module
    QEIVelocityConfigure(QEI0_BASE, QEI_VELDIV_1, 32000);
    //Enable the Velocity capture module
    QEIVelocityEnable(QEI0_BASE);
}

void Timer0IntHandler(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    direction = QEIDirectionGet(QEI0_BASE);
    position = QEIPositionGet(QEI0_BASE);

    //En_Read=(int16_t) position;
    if (position > pos_max/2)
    {
        En_Read = -(pos_max-position);
    } else
    {
        En_Read = position;
    }

    //velocity = QEIVelocityGet(QEI0_BASE);
    velocity = En_Read - xOld;
```

```
        //rpm = (40000*60*velocity)/32000/2;

        newDuty=(xDesired-En_Read)/Kp;          //calculate the new duty value;


        //newDuty = -velocity /Kd;    //calculate new duty value;
       // newDuty = 1;
        //newDuty=(errorSum+(xDesired-En_Read+errorOld)/2)/Ki;
            //newDuty=((xDesired-En_Read)/Kp)+velocity*direction/Kd;          //calculate thenew duty
value;


        if (newDuty < 0) {
              GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1 | GPIO_PIN_2, 2); // PD1
              newDuty = -newDuty;
        } else {
              GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1 | GPIO_PIN_2, 4); // PD2
        }
        if (newDuty > 100) {
              newDuty = 100;
        }
       widthAdjust = (ui32Load * newDuty) / 100;
        //widthAdjust=10;
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, widthAdjust+1);    // Set PWM Pulse Width

        errorSum += (xDesired - En_Read + errorOld) / 2;
        errorOld = xDesired - En_Read;
        count += 1;
```

```
        if (count > 2500) {
                xDesired = xMax - xDesired;
                errorSum = 0;
                count = 0;
        }
        xOld = En_Read;
}
```