FAMU- FSU College of Engineering

Department of Electrical and Computer Engineering

Fall 2024

EEL-4746L Microcontroller Based Systems Design Lab Report

Section No:	03
Lab Instructor:	Emil Lobachev
Lab No:	11
Lab Title:	Project : Joystick Motor Controller
Names:	Ruth Massock, Kenil Patel, Patrick Laciuga
Date Performed:	12/02/2024
Date Delivered:	12/02/2024

Contents

1. Introduction

In the lab, we designed a joystick-controlled stepper motor system using the MSP-430 Launchpad and the MKII Educational BoosterPack. The system interprets joystick inputs to control the motor's direction and speed, switching between different modes based on user interactions. This project aimed to deepen our understanding of real-time embedded systems by integrating hardware and software components. By completing this lab, we gained practical experience in motor control, joystick interfacing, and implementing debug features for testing and verification.

2. Design Requirements

This design utilizes the MSP-430 Launchpad and the MKII Educational BoosterPack to control a stepper motor based on joystick input. Upon powering up, the system enters standby mode, where the motor remains off, and LED2 flashes at 1 Hz to indicate readiness. The joystick's Y-axis controls clockwise motor rotation: a position above 75% triggers high-speed rotation, while below 25% initiates slow-speed rotation. If the Y-axis is centered, the X-axis governs counterclockwise rotation, with similar speed thresholds. When the joystick push button is pressed, the system switches to pattern mode, executing a predefined sequence of motor positions before resuming regular operation. Debug mode is activated by the MKII push buttons, overriding joystick inputs to manually control the motor's speed and direction or force it into pattern mode. LEDs provide real-time feedback for operation states, ensuring users can easily interpret the system's status. The design prioritizes efficient control logic, ensuring seamless transitions between modes and reliable motor operation.

3. Theoretical Design

Top-Level Design:

This design uses the MSP430FR5994 LaunchPad and MKII BoosterPack to control a stepper motor based on joystick inputs. The system has four main modes: standby, clockwise motion, counterclockwise motion, and pattern mode, each determined by the joystick's X and Y-axis positions or the pushbutton. LEDs on the LaunchPad indicate the active mode, while Timer A interrupts handle real-time motor step updates. Debug mode

allows manual control of motor direction and speed using push buttons S1 and S2. The motor operates in an eight-step sequence for smooth motion.

Pseudocode For Lab:

```
// Configure peripherals
Configure GPIO pins for LEDs and motor control
Initialize ADC for joystick input
Initialize push buttons for debug mode
Configure Timer A for motor control and interrupts
// Main loop
main()
begin
   Initialize system peripherals
   while(1)
   begin
      Joystick() // Read and interpret joystick inputs
      Motor_Controller() // Determine motor behavior based on inputs
      LED_Update() // Update LED states for mode indication
   end
end
// Timer A Interrupt Service Routine
Timer_A_ISR()
begin
   Motor_Driver() // Advance motor step sequence
end
// Joystick Handling
Joystick()
begin
   Read joystick X and Y positions
   Determine mode based on joystick input thresholds
end
// Debug Mode Handling
Debug()
begin
   If (S1 pressed and S2 not pressed)
      Set motor to forward high speed
   Else if (S2 pressed and S1 not pressed)
      Set motor to reverse low speed
   Else if (both S1 and S2 pressed)
      Enter pattern mode
```

```
end
// Pattern Mode
Pattern()
begin
   Cycle through predefined stepper motor positions
   Wait for completion of pattern before exiting mode
end
```

4. Synthesized Design

C Code:

```c
/*****************************************************************
*Author: Patrick Laciuga
* Co-Author:Ruth Massock
* Fall 2024
* Lab Section: 03
* Created: 12/02/2024
* Lab Number: Design Project
* Description: Main program for joystick-controlled stepper motor.
*****************************************************************/
#include "LcdDrivermsp430/Crystalfontz128x128_ST7735.h"
#include "LcdDrivermsp430/HAL_MSP_EXP430FR5994_Crystalfontz128x128_ST7735.h"
#include "grlib.h"
#include "driverlib.h"
#include <stdint.h>
#include <stdio.h>

// Global Variables

Graphics_Context g_sContext;
uint16_t JoyStickX, JoyStickY;
Timer_B_outputPWMParam MyTimerB;
Timer_A_initUpModeParam MyTimerA;

// Function Headers

void LCD_init(void);
void ADC_init(void);
void joyStick_init();
void configTimerA(uint16_t,uint16_t);
void myTimerADelay(uint16_t,uint16_t);
void configurePortIO(void);
void config_mkII_interrupts(void);
void myMotorDriver(void);
void myMotorController(void);
void handleJoystickInput(uint16_t,uint16_t);
void handleDebugMode(void);
void rotate30DegCW(void);
void newMotorController();
void patternExec(void);
void setupTimerB();
void startTimerB();
void stopTimerB();
typedef enum {OFFstby,CW,CCW, PATTERN} motorMode;
typedef enum {SLOW, NORMAL, FAST} motorSpeed;
motorSpeed speed;
motorMode motorTurn = OFFstby;
uint16_t delayMotor;
uint8_t PBS1,PBJ1;
uint8_t PBS2,PBS3;

#define YHIGH 0xA0B
#define YLOW 0x638
#define XHIGH 0xA0B
#define XLOW 0x638
#define XCENTER 0x763
#define YCENTER 0x763

uint16_t JoyStickX, JoyStickY;
```

```
// Global variable to track LED state
volatile int joystickState = 0;
volatile int joystickStateled=1;
volatile uint16_t ii=0;
enum {ON,OFF}joystickMode=OFF;
void toggle(void);


uint8_t MotorState = 0;


uint8_t motorSeq; // Global variable to hold the motor sequence step (0-7)

// Function prototypes

void myMotorDriver(void);
char buffer[100];
// Main function

void main(void) {
    // Stop watchdog timer
    WDT_A_hold(WDT_A_BASE);

   // Set up the GPIO for each LED based on motor sequence
        //Stop WDT
        WDT_A_hold(WDT_A_BASE);

        // Initialize Joystick
        joyStick_init();

        // Configure

        configurePortIO();

        // Activate Configuration
        PMM_unlockLPM5();

        // Initialize ADC
        ADC_init();

        // Initialize LCD
        LCD_init();

        setupTimerB();

        MotorState = 0;
        joystickMode=OFF;

        GPIO_clearInterrupt(GPIO_PORT_P4, GPIO_PIN3);

        // Configure Timer A for interrupts
        configTimerA(1000,TIMER_A_CLOCKSOURCE_DIVIDER_2);// Configure the timer parameters
        Timer_A_initUpMode(TIMER_A0_BASE,&MyTimerA);// Enable Local Interrupts
        Timer_A_enableInterrupt(TIMER_A0_BASE);
        __enable_interrupt();// Enable Global Interrupts
        while(1)
        {
            handleDebugMode();
```

```
            if(speed==FAST)
            {
                delayMotor=1562;
            } else
            {
                delayMotor=3125;
            }

            __disable_interrupt();
            Timer_A_disableInterrupt(TIMER_A0_BASE);
            configTimerA(delayMotor,TIMER_A_CLOCKSOURCE_DIVIDER_2);// Configure the timer parameters
            Timer_A_initUpMode(TIMER_A0_BASE,&MyTimerA);// Enable Local Interrupts
            Timer_A_enableInterrupt(TIMER_A0_BASE);
            __enable_interrupt();// Enable Global Interrupts
            Timer_A_startCounter(TIMER_A0_BASE,TIMER_A_UP_MODE);
            __low_power_mode_0();
            __no_operation();
        }
}


//motor ISR
//FOR motorController
#pragma vector=TIMER0_A1_VECTOR

__interrupt void motorISR(void){
    newMotorController();
    // Clear Timer Interrupt Flag
    Timer_A_clearTimerInterrupt(TIMER_A0_BASE);
    Timer_A_stop(TIMER_A0_BASE);
    __low_power_mode_off_on_exit();
}


// LED ISR
//To Handle Toggle
#pragma vector = TIMER0_B0_VECTOR
__interrupt void TimerBISR(void){
    GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
    GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN1);
}


//Debug Function
// Function to handle debug mode

void handleDebugMode() {
    motorTurn = OFFstby;
    // Read push buttons states (e.g., from GPIO)
    PBS1 = GPIO_getInputPinValue(GPIO_PORT_P5, GPIO_PIN6);
    PBS2 = GPIO_getInputPinValue(GPIO_PORT_P5, GPIO_PIN5);
    PBS3= GPIO_getInputPinValue(GPIO_PORT_P6, GPIO_PIN2);//joybutton

    if ((!PBS1) && (!PBS2)) {
        GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
        GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
        // Both buttons pressed, enter pattern mode
        joystickMode=OFF;
```

```
175
176         MotorState = 0;
177         motorTurn = PATTERN;
178
179
180     } else if (!PBS1) {
181         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
182         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
183
184         joystickMode=OFF;
185         motorTurn = CW;
186         speed = FAST;
187
188
189
190
191     } else if (!PBS2) {
192         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
193         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
194
195
196         joystickMode=OFF;
197         motorTurn = CCW;
198         speed = SLOW;
199
200
201     } else
202     {
203
204         joystickMode=ON;
205
206         ADC12_B_startConversion(ADC12_B_BASE, ADC12_B_START_AT_ADC12MEM0, ADC12_B_SEQOFCHANNELS);
207
208         while (ADC12_B_getInterruptStatus(ADC12_B_BASE, 0, ADC12_B_IFG1) != ADC12_B_IFG1);
209
210             JoyStickY = ADC12_B_getResults(ADC12_B_BASE, ADC12_B_MEMORY_1);
211             JoyStickX = ADC12_B_getResults(ADC12_B_BASE, ADC12_B_MEMORY_0);
212
213             ADC12_B_clearInterrupt(ADC12_B_BASE, 0, ADC12_B_IFG1);
214                      if (JoyStickY > (YHIGH)){
215                          GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
216                          GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN1);
217                             joystickState = 1;
218                             motorTurn = CW;
219                             speed = FAST;
220                         }
221                       else if (JoyStickY < YLOW)
222                     {
223                         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
224                         GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN1);
225                         joystickState = 2; // Y-axis low
226                         motorTurn = CW;
227                         speed = SLOW;
228                     }
229                     else if (JoyStickX > (XHIGH ))
230                     {
231                         GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN0);
232                         GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
```

```
                    joystickState = 3; // X-axis high
                    motorTurn = CCW;
                    speed = FAST;
                }
                else if (JoyStickX < XLOW)
                {
                    GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN0);
                    GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
                    joystickState = 4; // X-axis low
                    motorTurn = CCW;
                    speed = SLOW;
                }
                    else if (!PBS3)
                    {
                        GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN0);
                        GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN1);
                        joystickState = 5;
                        motorTurn = PATTERN;
                    }
                else {
                    joystickState = OFFstby;

            }
    }


}
//ConfigIO Function
//Configure pins
void configurePortIO() {

        GPIO_setAsOutputPin(GPIO_PORT_P3,GPIO_PIN7);
        GPIO_setAsOutputPin(GPIO_PORT_P3,GPIO_PIN6);
        GPIO_setAsOutputPin(GPIO_PORT_P3,GPIO_PIN5);
        GPIO_setAsOutputPin(GPIO_PORT_P3,GPIO_PIN4);
        GPIO_setAsOutputPin(GPIO_PORT_P1,GPIO_PIN1);
        GPIO_setAsOutputPin(GPIO_PORT_P1,GPIO_PIN0);



        // Set input pins
        GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5,GPIO_PIN6);
        GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5,GPIO_PIN5);
        GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P6, GPIO_PIN2);  // jp2




        //Turn Off all Pins

        GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN7);
        GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN6);
        GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN5);
        GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN4);
        GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
        GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);

```

```
291
292 }
293
294
295 // Function to drive the LEDs based on the motor sequence
296 void myMotorDriver()  {
297
298     switch (MotorState) {
299         case 0:
300             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
301             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
302             //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
303             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
304             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
305             break;
306
307         case 1:
308             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
309             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
310           //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
311             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
312             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
313             break;
314
315         case 2:
316             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
317             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
318           //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
319             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
320             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
321             break;
322
323         case 3:
324             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
325             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
326           //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
327             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
328             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
329             break;
330
331         case 4:
332             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
333             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
334           //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
335             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
336             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
337             break;
338
339         case 5:
340             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
341             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
342           //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
343             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
344             GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
345             break;
346
347         case 6:
348             GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
```

```
            GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
          //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
            GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
            GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
            break;

        case 7:
            GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // A
            GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4); //A_
          //  GPIO_setOutputHighOnPin(GPIO_PORT_J4, GPIO_PIN40); // mirror
            GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // B
            GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); // B_
            break;

        default:
            // A RED LP
            GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN7);
            // B RED EB
            GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN6);
            // ABAR BLUE EB
            GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN5);
            // BBAR GREEN EB
            GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN4);
            break;
    }
}



void joyStick_init(){
    // JoyStick X
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3, GPIO_PIN3, GPIO_TERNARY_MODULE_FUNCTION);

    // JoyStick Y
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1, GPIO_PIN2, GPIO_TERNARY_MODULE_FUNCTION);
}

// LCD_Init
// Configures mkII LCD display
// Inputs: none
// Returns: none

void LCD_init()
{
/* Initializes display */
Crystalfontz128x128_Init();
/* Set default screen orientation */
Crystalfontz128x128_SetOrientation(0);
/* Initializes graphics context */
Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
Graphics_clearDisplay(&g_sContext);
}

// ADC_init
```

```
// Configures ADC to use joystick inputs
// Inputs: none
// Returns: none

void ADC_init(){
    //Initialize the ADC12B Module
    /*
    * Base address of ADC12B Module
    * Use internal ADC12B bit as sample/hold signal to start conversion
    * USE MODOSC 5MHZ Digital Oscillator as clock source
    * Use default clock divider/pre-divider of 1
    * Not use internal channel
    */
    ADC12_B_initParam initParam = {0};
    initParam.sampleHoldSignalSourceSelect = ADC12_B_SAMPLEHOLDSOURCE_SC;
    initParam.clockSourceSelect = ADC12_B_CLOCKSOURCE_ADC12OSC;
    initParam.clockSourceDivider = ADC12_B_CLOCKDIVIDER_1;
    initParam.clockSourcePredivider = ADC12_B_CLOCKPREDIVIDER__1;
    initParam.internalChannelMap = ADC12_B_NOINTCH;
    ADC12_B_init(ADC12_B_BASE, &initParam);

    //Enable the ADC12B module
    ADC12_B_enable(ADC12_B_BASE);
    /*
    * Base address of ADC12B Module
    * For memory buffers 0-7 sample/hold for 64 clock cycles
    * For memory buffers 8-15 sample/hold for 4 clock cycles (default)
    * Enable Multiple Sampling
    */
    ADC12_B_setupSamplingTimer(ADC12_B_BASE,
      ADC12_B_CYCLEHOLD_16_CYCLES,
      ADC12_B_CYCLEHOLD_4_CYCLES,
      ADC12_B_MULTIPLESAMPLESENABLE);
    //Configure Memory Buffer
    /*
    * Base address of the ADC12B Module
    * Configure memory buffer 0
    * Map input A1 to memory buffer 0
    * Vref+ = AVcc
    * Vref- = AVss
    * Memory buffer 0 is not the end of a sequence
    */
    //  JoyStickXParam Structure

    ADC12_B_configureMemoryParam joyStickXParam = {0};
    joyStickXParam.memoryBufferControlIndex = ADC12_B_MEMORY_0;
    joyStickXParam.inputSourceSelect = ADC12_B_INPUT_A2;
    joyStickXParam.refVoltageSourceSelect = ADC12_B_VREFPOS_AVCC_VREFNEG_VSS;
    joyStickXParam.endOfSequence = ADC12_B_NOTENDOFSEQUENCE;
    joyStickXParam.windowComparatorSelect = ADC12_B_WINDOW_COMPARATOR_DISABLE;
    joyStickXParam.differentialModeSelect = ADC12_B_DIFFERENTIAL_MODE_DISABLE;
    ADC12_B_configureMemory(ADC12_B_BASE, &joyStickXParam);

    //  JoyStickYParam Structure
    ADC12_B_configureMemoryParam joyStickYParam = {0};
    joyStickYParam.memoryBufferControlIndex = ADC12_B_MEMORY_1;
    joyStickYParam.inputSourceSelect = ADC12_B_INPUT_A15;
    joyStickYParam.refVoltageSourceSelect = ADC12_B_VREFPOS_AVCC_VREFNEG_VSS;
```

```
    joyStickYParam.endOfSequence = ADC12_B_ENDOFSEQUENCE;
    joyStickYParam.windowComparatorSelect = ADC12_B_WINDOW_COMPARATOR_DISABLE;
    joyStickYParam.differentialModeSelect = ADC12_B_DIFFERENTIAL_MODE_DISABLE;
    ADC12_B_configureMemory(ADC12_B_BASE, &joyStickYParam);

    // Clear Interrupt
    ADC12_B_clearInterrupt(ADC12_B_BASE,0,ADC12_B_IFG1);


}

// configTimerA
// Configuration Parameters for TimerA
// Inputs: delayValue -- number of count cycles
//         clockDividerValue -- clock divider
// Returns: None
void configTimerA(uint16_t delayValue, uint16_t clockDividerValue)
{
    MyTimerA.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    MyTimerA.clockSourceDivider = clockDividerValue;
    MyTimerA.timerPeriod = delayValue;
    MyTimerA.timerClear = TIMER_A_DO_CLEAR;
    MyTimerA.startTimer = false;
}

void rotate30DegCW(void)
{
    int i=0;
    for(i=0; i<33; i++)
    {
        MotorState++;
        if(MotorState>7)
        {
            MotorState=0;
        }
        myMotorDriver();
         __delay_cycles(5000);
    }


}

void patternExec(void)
{
    MotorState = 0;
    myMotorDriver();
    //Using Pattern 3.
    //12 to 1
    rotate30DegCW();
     __delay_cycles(500000);
    //1 to 7
    int i=0;
    for(i=0;i<6;i++)
    {
        rotate30DegCW();
    }
     __delay_cycles(500000);
```

```
523     //7 to 10
524     i=0;
525     for(i=0;i<3;i++)
526     {
527         rotate30DegCW();
528     }
529      __delay_cycles(500000);
530     //10 to 4
531     i=0;
532     for(i=0;i<6;i++)
533     {
534         rotate30DegCW();
535     }
536      __delay_cycles(500000);
537     //4 to 8
538     i=0;
539     for(i=0;i<4;i++)
540     {
541         rotate30DegCW();
542     }
543      __delay_cycles(500000);
544     //8 to 2
545     i=0;
546     for(i=0;i<6;i++)
547     {
548         rotate30DegCW();
549     }
550      __delay_cycles(500000);
551     //2 to 10
552     i=0;
553     for(i=0;i<8;i++)
554     {
555         rotate30DegCW();
556     }
557      __delay_cycles(500000);
558     //10 to 7
559     i=0;
560     for(i=0;i<9;i++)
561     {
562         rotate30DegCW();
563     }
564      __delay_cycles(500000);
565     //7 to 6
566     i=0;
567     for(i=0;i<11;i++)
568     {
569         rotate30DegCW();
570     }
571     __delay_cycles(500000);
572     //6 to 12
573     i=0;
574     for(i=0;i<6;i++)
575     {
576         rotate30DegCW();
577     }
578     MotorState = 0;
579     myMotorDriver();
580 }
```

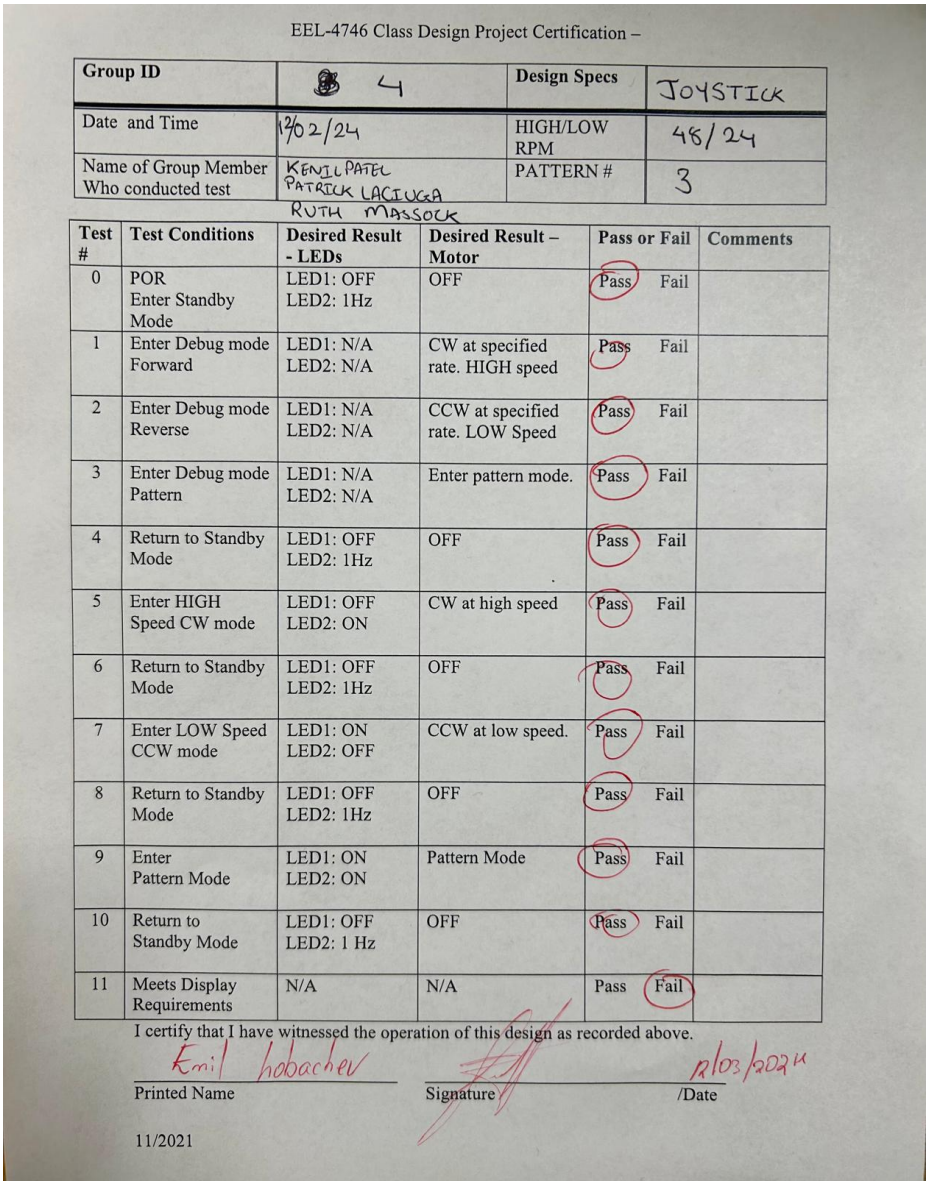
```
581
582 void myTimerADelay(uint16_t delayValue, uint16_t clockDividerValue)
583 {
584
585    configTimerA(delayValue,clockDividerValue);  // Configure the timer parameters
586    Timer_A_initUpMode(TIMER_A0_BASE,&MyTimerA); // Initialize the timer
587    Timer_A_startCounter(TIMER_A0_BASE,TIMER_A_UP_MODE);  // Start Timer
588    while((TA0CTL&TAIFG) == 0);                   // Wait for TAIFG to become Set
589    Timer_A_stop(TIMER_A0_BASE);                  // Stop timer
590    Timer_A_clearTimerInterrupt(TIMER_A0_BASE);   // Reset TAIFG to Zero
591 }
592
593
594 void newMotorController(){
595     switch(motorTurn){
596
597     case OFFstby:
598         startTimerB();
599         break;
600     case CW:
601         stopTimerB();
602         if(MotorState < 7) MotorState++;else MotorState = 0;
603
604         break;
605     case CCW:
606         stopTimerB();
607         if(MotorState >0) MotorState--;else MotorState = 7;
608
609
610         break;
611     case PATTERN:
612         patternExec();
613         break;
614
615     default:
616        MotorState = 0;
617        break;
618     }
619
620     myMotorDriver();
621
622     }
623
624
625 void setupTimerB(){
626     TB0CCTL0 = CCIE;
627     TB0CCR0 = 32767;
628     TB0CTL = TBSSEL_1 | MC_1 | TBCLR;
629 }
630
631 void startTimerB(){
632     TB0CTL |= MC_1;
633 }
634
635 void stopTimerB(){
636     TB0CTL &= ~MC_3;
637 }
638
```

5. Experimental Results

During the experimental phase, we conducted multiple tests to validate the functionality of the joystick-controlled stepper motor. Out of the 11 required tests, 10 were successfully passed. Each mode outlined in the design requirements was verified for accuracy, including standby, clockwise, counterclockwise, pattern, and debug modes. The motor responded accurately to joystick inputs, and the LEDs provided appropriate feedback for each mode. However, one test failed due to an oversight in configuring the display text on the LCD screen. This issue occurred because we neglected to implement the required logic for LCD text output, which we realized during

the certification process. Despite this, the motor control and mode functionality performed reliably, meeting the primary objectives of the project.

Certification sheet:



EEL-4746 Class Design Project Certification –

Group ID	4	Design Specs	JOYSTICK
Date and Time	12/02/24	HIGH/LOW RPM	48/24
Name of Group Member Who conducted test	KENIL PATEL PATRICK LACIUGA RUTH MASSOCK	PATTERN #	3

Test #	Test Conditions	Desired Result - LEDs	Desired Result – Motor	Pass or Fail	Comments
0	POR Enter Standby Mode	LED1: OFF LED2: 1Hz	OFF	Pass Fail	
1	Enter Debug mode Forward	LED1: N/A LED2: N/A	CW at specified rate. HIGH speed	Pass Fail	
2	Enter Debug mode Reverse	LED1: N/A LED2: N/A	CCW at specified rate. LOW Speed	Pass Fail	
3	Enter Debug mode Pattern	LED1: N/A LED2: N/A	Enter pattern mode.	Pass Fail	
4	Return to Standby Mode	LED1: OFF LED2: 1Hz	OFF	Pass Fail	
5	Enter HIGH Speed CW mode	LED1: OFF LED2: ON	CW at high speed	Pass Fail	
6	Return to Standby Mode	LED1: OFF LED2: 1Hz	OFF	Pass Fail	
7	Enter LOW Speed CCW mode	LED1: ON LED2: OFF	CCW at low speed.	Pass Fail	
8	Return to Standby Mode	LED1: OFF LED2: 1Hz	OFF	Pass Fail	
9	Enter Pattern Mode	LED1: ON LED2: ON	Pattern Mode	Pass Fail	
10	Return to Standby Mode	LED1: OFF LED2: 1 Hz	OFF	Pass Fail	
11	Meets Display Requirements	N/A	N/A	Pass Fail	

I certify that I have witnessed the operation of this design as recorded above.

Emil Hobachev — Printed Name

Signature

12/03/2024 — /Date

11/2021

6. Summary

The design achieved 10 out of 11 successful tests, resulting in approximately 91% compliance with the specified requirements. While the core motor control functionality and mode-based behavior met expectations, the omission of LCD text output functionality was a noted failure. If given the opportunity to modify the design, we would address this issue by incorporating and thoroughly testing the code for LCD text output to ensure complete functionality. Despite this shortfall, the design demonstrated its reliability in all critical aspects of motor control and user interaction.

7. Lessons Learned

Through this project, we gained practical experience in embedded system design and stepper motor control. First, we developed a deeper understanding of interfacing the MSP-430 microcontroller with peripheral devices, including stepper motors and joysticks. Second, we learned the importance of systematically addressing all design requirements, as the omission of the LCD text output highlighted the need for meticulous attention to detail. Third, we reinforced our skills in debugging and testing to ensure robust system performance. If we were to repeat this lab, we would allocate additional time for testing secondary features like LCD text output and ensure that all requirements are thoroughly reviewed and implemented.