CECS 447 - Spring 2024 - Project 5 (FINAL)

I2C Bus Communication

By

Edward Ramos – Maryith Garzon - Rod Agatep (GROUP 9)

17 April 2024

Leverage I2C bus communication with three distinct external I2C components.

Introduction

The purpose of this project is to design, code, and build an embedded system that leverages I2C bus communication to configure and retrieve data from three distinct external I2C components: a TCS34725, MPU6050, and 16x2 LCD. Functions include identifying colors, changing the angle of a servo motor, and displaying color and angle.

Operation

Video Links: https://youtu.be/Brs7ijDwMFw

This program runs on a TM4C123G ARM Cortex Microcontroller; the code is written on Keil uVision5 and is downloaded to the board using the included USB cable.

This project consists of six modules. The first module is the 32-bit wide WTIMER0, is used to generate a 1ms timing. The second module is UART, this section will be able to detect using ID register, configure specific settings, and work with LED and UART terminal. The third module is a color RBG sensor for the TCS34725, with same functions as UART module. The fourth module is an MPU6050 6-dof IMU, with same 3 functions as UART and TCS. The fifth module is a 16x2 LCD. Able to display a first and last name. And the final module is to move an angular servo to the following degrees: 0, -45, 0, 45, 90, 0, 90, 0.

The final submission and demo will consist of all these modules in a full system: the system will identify the color of an RBG strip connected to the servo, the servo's angle is adjusted by the MPU6050 and the user. While the LCD provide real-time info updates, showing the color detected and angle of the MPU.

Theory

TM4C Launchpad

The microcontroller platform that serves as the base of this project.

16x2 LCD with I2C Interface

LCD screen used to display the color and angle detected. Connected to the I2C Interface

I2C Interface

Connected to the 16x2 LCD.

TCS34725 RBG Color Sensor

Detect the color of the RGB strip.

MPU6050 6-degree-of-freedom IMU

Detects movement of device.

Angular Servo Motor

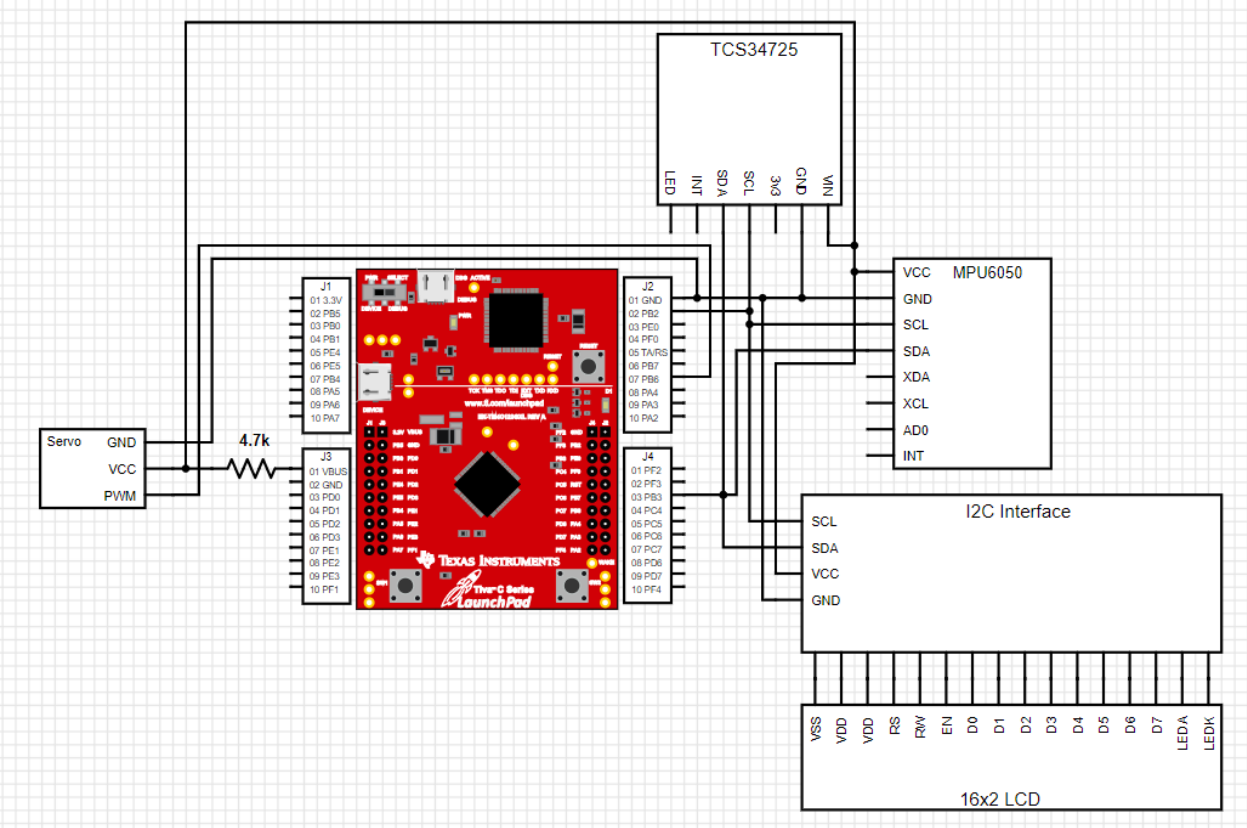
Moves to a specific angle based on the MPU6050.

Tera Term

Serial Terminal Application. Used to run system.

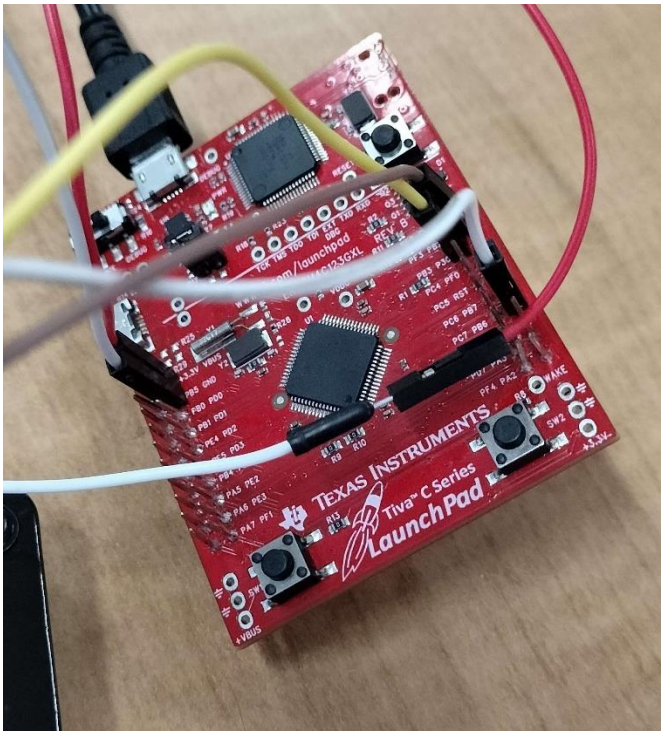
Hardware Design

SCHEMATIC

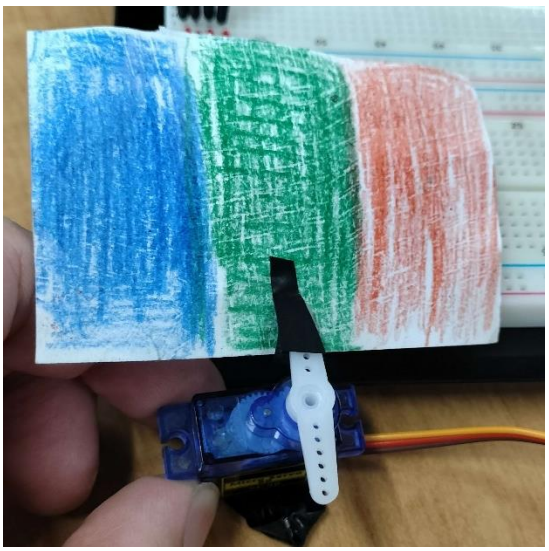


LAUNCHPAD AND CIRCUIT

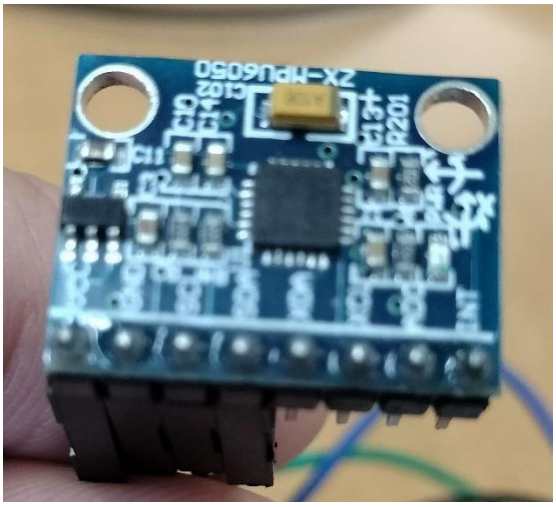
LAUNCHPAD



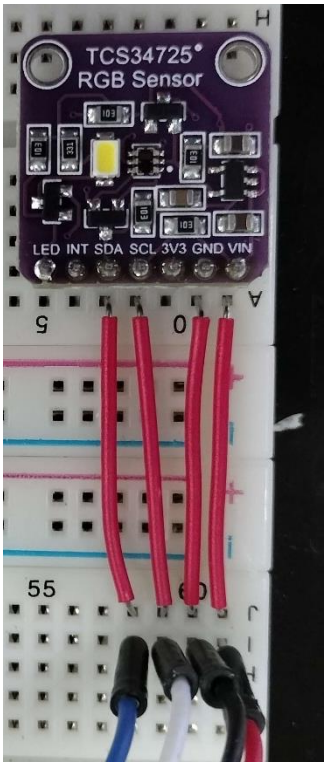
SERVO



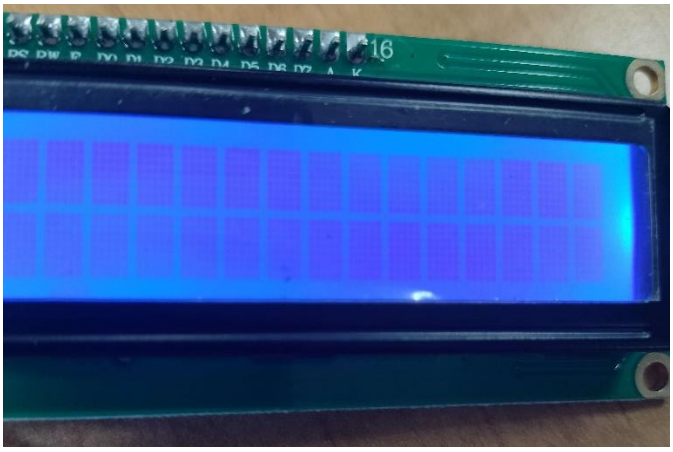
MPU6050

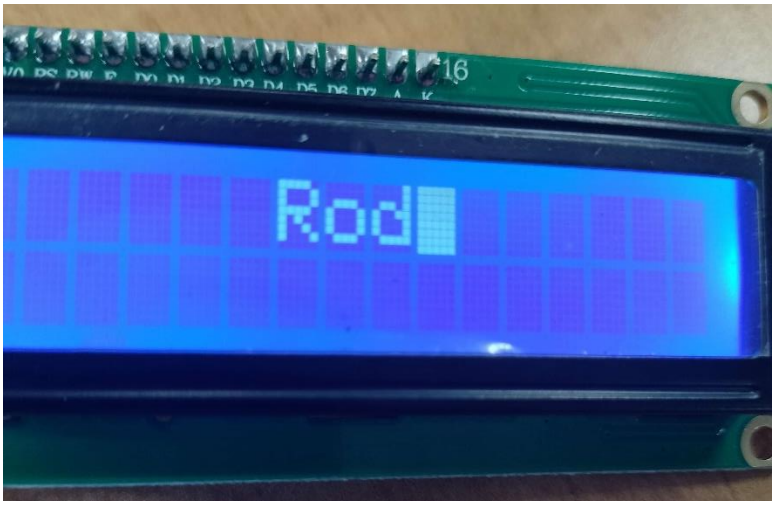


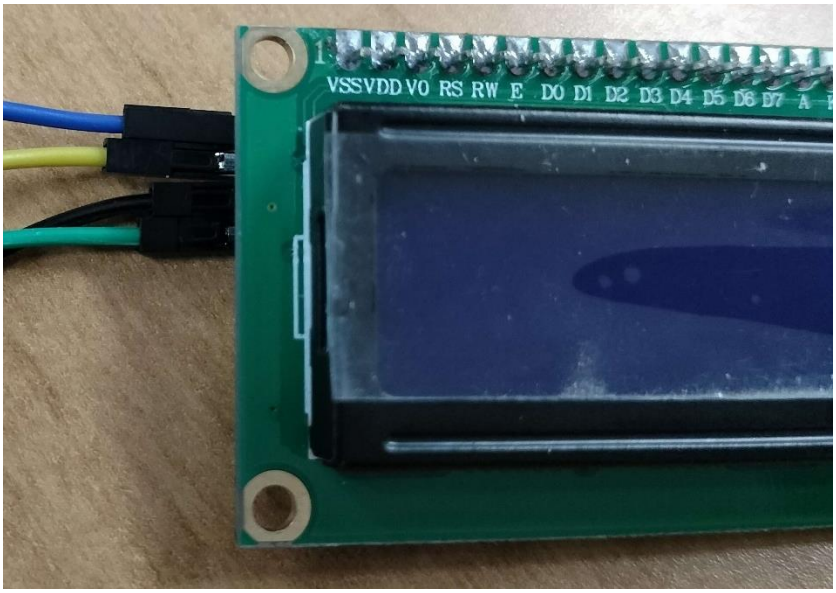
TCS



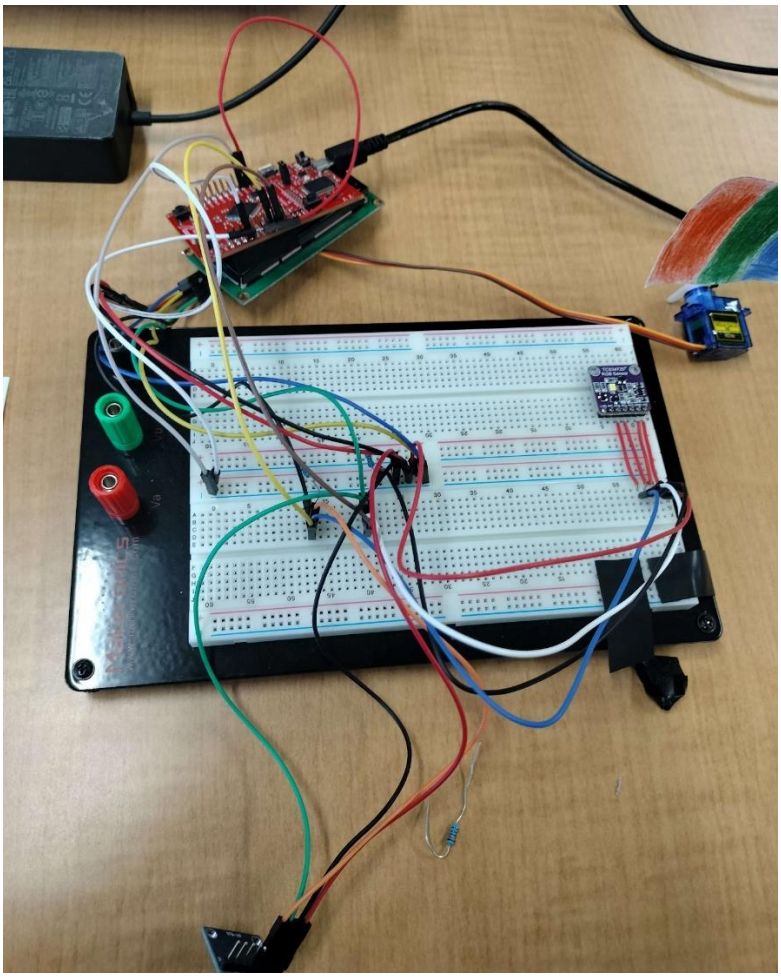
LCD







FULL SYSTEM



SOURCE CODE

I2CTestMain.c

```c
/*
 * I2CTestMain.c
 *
 *  Test the basic functionality of I2C by reading and writing to the
 *  TCS34727 RGB Sensor and MPU6050 6-dof IMU, as well as controlling
 *  a Servo Motor and a 16x2 LCD
 *
 * Created on: May 24th, 2023
 *     Author: Jackie Huynh
 *
 */

#include "tm4c123gh6pm.h"
#include "I2C.h"
#include "UART0.h"
#include "TCS34727.h"
#include "MPU6050.h"
#include "ButtonLED.h"
#include "util.h"
#include "Servo.h"
#include "LCD.h"
#include <stdio.h>
#include <string.h>
#include "ModuleTest.h"

/* List of Predefined Macros for individual Peripheral Testing */
//#define DELAY
//#define UART
//#define I2C
//#define TCS34727
//#define MPU6050
//#define SERVO
#define LCD
//#define FULL_SYSTEM

int main(void){

  /* Peripheral Initialization */
  UART0_Init();
  LED_Init();
  BTN_Init();

  #if defined(DELAY) || defined(TCS34727) || defined(MPU6050) || defined(LCD) || defined(FULL_SYSTEM)
  WTIMER0_Init();
  #endif

  #if defined (I2C) || defined(TCS34727) || defined(MPU6050) || defined(LCD) || defined(FULL_SYSTEM)
  I2C0_Init();
  #endif
```

```
51  #if defined(TCS34727) || defined(FULL_SYSTEM)
52  /* Color Sensor Initialization */
53  TCS34727_Init();
54  #endif
55
56  #if defined(MPU6050) || defined(FULL_SYSTEM)
57  /* MPU6050 Initialization */
58  MPU6050_Init();
59  #endif
60
61  #if defined(SERVO) || defined(FULL_SYSTEM)
62  /* Servo Initialization */
63  Servo_Init();
64  WTIMER0_Init();
65  #endif
66
67  #if defined(LCD) || defined(FULL_SYSTEM)
68  /* LCD Initialization */
69  LCD_Init();
70  #endif
71
72  while(1){
73
74    #ifdef DELAY
75    Module_Test(DELAY_TEST);
76    #endif
77
78    #ifdef UART
79    Module_Test(UART_TEST);
80    #endif
```

```
81
82    #ifdef I2C
83    Module_Test(I2C_TEST);
84    #endif
85
86    #ifdef MPU6050
87    Module_Test(MPU6050_TEST);
88    #endif
89
90    #ifdef TCS34727
91    Module_Test(TCS34727_TEST);
92    #endif
93
94    #ifdef SERVO
95    Module_Test(SERVO_TEST);
96    #endif
97
98    #ifdef LCD
99    Module_Test(LCD_TEST);
100   #endif
101
102   #ifdef FULL_SYSTEM
103   Module_Test(FULL_SYSTEM_TEST);
104   #endif
105
106 }
107
108 return 0;
109 }
110
```

I2C.c

```c
/*
 * I2C.h
 *
 *  Main implementation of the I2C Init, Read, and Write Function
 *
 * Created on: May 24th, 2023
 *    Author: Jackie Huynh
 *
 */

#include "I2C.h"
#include "tm4c123gh6pm.h"

/*
 *  -------------------I2C0_Init------------------
 *  Basic I2C Initialization function for master mode @ 100kHz
 *  Input: None
 *  Output: None
 */
void I2C0_Init(void){

  /* Enable Required System Clock */
  SYSCTL_RCGC1_R |= EN_I2C0_CLOCK;              //Enable I2C0 System Clock
  SYSCTL_RCGC2_R |= EN_GPIOB_CLOCK;             //Enable GPIOB System Clock

  //Wait Until GPIOx System Clock is enabled
  while((SYSCTL_RCGC2_R&EN_GPIOB_CLOCK)!= EN_GPIOB_CLOCK);

  /* GPIOx I2C Alternate Function Setup */
  GPIO_PORTB_DEN_R   |= I2C0_PINS;               //Enable Digital I/O
  GPIO_PORTB_AFSEL_R |= I2C0_PINS;               //Enable Alternate Function Selection

  //Select I2C0 as the alternate function
  GPIO_PORTB_PCTL_R  |= (GPIO_PORTB_PCTL_R&I2C0_ALT_FUNC_SET)+I2C0_ALT_FUNC_SET;
  GPIO_PORTB_ODR_R   |= I2C0_SDA_PIN;           //Enable Open Drain for SDA pin
  GPIO_PORTB_AMSEL_R &= I2C0_PINS;               //Disable Analog Mode

  /*  I2C0 Setup as Master Mode @ 100kBits  */
  I2C0_MCR_R |= EN_I2C0_MASTER;                 //Configure I2C0 as Master

  /* Configuring I2C Clock Frequency to 100KHz

    TPR = (System Clock / (2*(SCL_LP + SCL_HP) * SCL_CLK)) - 1
    SCL_LP and SCL_HP are fixed
    SCL_LP = 6 & SCL_HP = 4

    Example if we want to configure I2C speed to 100kHz for 40MHz system clock
    TPR = (40MHz / ((2*(6+4)) * 100kHz)) - 1    (Convert Everything to Hz)
    TPR = 19

    TPR = 7

  */

  //Write the I2CMTPR register with the value of 0x0000.0003.

  I2C0_MTPR_R = I2C0_CLOCK_VALUE;

}

/*
 *  -------------------I2C0_Receive------------------
 *  Polls to receive data from specified peripheral
 *  Input: Slave address & Slave Register Address
 *  Output: Returns 8-bit data that has been received
 */
uint8_t I2C0_Receive(uint8_t slave_addr, uint8_t slave_reg_addr){

  char error;                                   //Temp Variable to hold errors

  /* Check if I2C0 is busy: check MCS register Busy bit */
  while(I2C0_MCS_R&I2C_MCS_BUSY);

  /* Configure I2C0 Slave Address and Read Mode */
  I2C0_MSA_R = (slave_addr<<1);                 // Slave Address is the 7 MSB
  I2C0_MDR_R = slave_reg_addr;                  // Set Data Register to slave register address

  /* Initiate I2C by generating a START & RUN cmd:
     Set MCS register START bit to generate and RUN bit to enable I2C Master
  */
  I2C0_MCS_R = I2C_MCS_START | I2C_MCS_RUN;

  /* Wait until write is done: check MCS register to see is I2C is still busy */
  while(I2C0_MCS_R&I2C_MCS_BUSY);

  /* Set I2C to Receive with Slave Address and change to Read */
  I2C0_MSA_R = (slave_addr<<1)|I2C0_READ;

  /* Initiate I2C by generating a repeated START, STOP, & RUN cmd */
  I2C0_MCS_R = I2C_MCS_START | I2C_MCS_STOP | I2C_MCS_RUN;

  /* Wait until I2C bus is not busy: check MCS register for I2C bus busy bit */
  while(I2C0_MCS_R&I2C_MCS_BUSBSY);

  /* Check for any error: read the error flag from MCS register */
```

```
  error = I2C0_MCS_R&I2C_MCS_ERROR;
  if(error != 0)
    return error;
  else
    return (I2C0_MDR_R&I2C_MDR_DATA_M);  // return I2C data register least significant 8 bits.

}

/*
 *  -------------------I2C0_Transmit------------------
 *  Transmit a byte of data to specified peripheral
 *  Input: Slave address, Slave Register Address, Data to Transmit
 *  Output: Any Errors if detected, otherwise 0
 */
uint8_t I2C0_Transmit(uint8_t slave_addr, uint8_t slave_reg_addr, uint8_t data){

  char error;                                 //Temp Variable to hold errors

  /* Check if I2C0 is busy: check MCS register Busy bit */
  while(I2C0_MCS_R&I2C_MCS_BUSY);

  /* Configure I2C Slave Address, R/W Mode, and what to transmit */
  I2C0_MSA_R = slave_addr<<1;               //Slave Address is the first 7 MSB
  I2C0_MSA_R &= ~I2C0_RW_PIN;              // Clear LSB to write
  I2C0_MDR_R = slave_reg_addr;                //Transmit register addr to interact

  /* Initiate I2C by generate a START bit and RUN cmd */
  I2C0_MCS_R = I2C_MCS_START | I2C_MCS_RUN;

  /* Wait until write has been completed */
```

```
  while(I2C0_MCS_R&I2C_MCS_BUSY);

  /* Update Data Register with data to be transmitted */
  I2C0_MDR_R =data;

  /* Initiate I2C by generating a STOP & RUN cmd */
  I2C0_MCS_R = I2C_MCS_STOP | I2C_MCS_RUN;

  /* Wait until write has been completed: check MCS register Busy bit */
  while(I2C0_MCS_R&I2C_MCS_BUSY);

  /* Wait until bus isn't busy: check MCS register for I2C bus busy bit */
  while(I2C0_MCS_R&I2C_MCS_BUSBSY);

  /* Check for any error: read the error flag from MCS register */
  error = I2C0_MCS_R&I2C_MCS_ERROR;
  if(error != 0)
    return error;
  else
    return 0;
}

//Has Yet to be Implemented
/*
 *  ----------------I2C0_Burst_Receive-----------------
 *  Polls to receive multiple bytes of data from specified
 *  peripheral by incrementing starting slave register address
 *  Input: Slave address, Slave Register Address, Data Buffer
 *  Output: None
 */
```

```
156 void I2C0_Burst_Receive(uint8_t slave_addr, uint8_t slave_reg_addr, uint8_t* data, uint32_t size){
157
158 }
159
160 /*
161  *  ----------------I2C0_Burst_Transmit-----------------
162  *  Transmit multiple bytes of data to specified peripheral
163  *  by incrementing starting slave address
164  *  Input: Slave address, Slave Register Address, Data Buffer to transmit
165  *  Output: None
166  */
167 uint8_t I2C0_Burst_Transmit(uint8_t slave_addr, uint8_t slave_reg_addr, uint8_t* data, uint32_t size){
168
169   char error;                           //Temp Error Variable
170
171   /* Asserting Param */
172   if(size <= 0)
173     return 0;
174
175   /* Check if I2C0 is busy */
176   while(I2C0_MCS_R & I2C_MCS_BUSY);
177
178   /* Configure I2C Slave Address, R/W Mode, and what to transmit */
179   I2C0_MSA_R = (slave_addr<<1);           //Slave Address is the first 7 MSB
180   I2C0_MSA_R &= ~I2C0_RW_PIN;            //Clear LSB to write
181   I2C0_MDR_R = slave_reg_addr;            //Transmit register addr to interact
182
183   /* Initiate I2C by generate a START bit and RUN cmd */
184   I2C0_MCS_R = I2C_MCS_START | I2C_MCS_RUN;
185
```

```
186   /* Wait until write has been completed */
187   while(I2C0_MCS_R & I2C_MCS_BUSY);
188
189   /* Loop to Burst Transmit what is stored in data buffer */
190   while(size > 1){
191
192     I2C0_MDR_R = *data++;             //Deference Pointer from data array and load into data reg. Post-Increment the pointer after
193     I2C0_MCS_R = RUN_CMD;                 //Initiate I2C RUN CMD
194     while((I2C0_MCS_R & I2C_MCS_BUSY));           //Wait until transmit is complete
195     size--;                               //Reduce size until 1 is left
196
197   }
198
199   I2C0_MDR_R = *data;              //Deference Pointer from data array and load into data reg
200   I2C0_MCS_R = I2C_MCS_STOP | I2C_MCS_RUN;            //Initiate I2C STOP condition and RUN CMD
201
202   /* Wait until write has been completed */
203   while(I2C0_MCS_R & I2C_MCS_BUSY);
204
205   /* Wait until bus isn't busy */
206   while(I2C0_MCS_R & I2C_MCS_BUSBSY);
207
208   /* Check for any error */
209   error = I2C0_MCS_R & I2C_MCS_ERROR;
210   if(error != 0)
211     return error;
212   else
213     return 0;
214 }
```

UART0.c

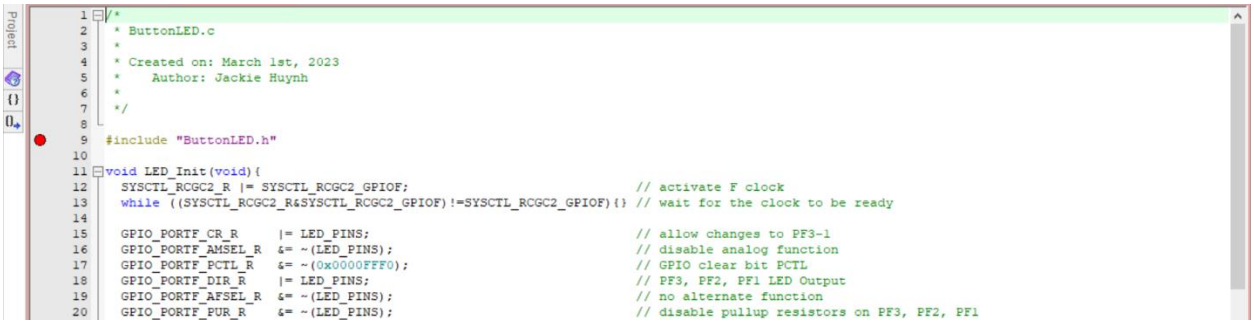
```
1  // UART.c
2  // Runs on TM4C123
3
4  // U0Rx (VCP receive) connected to PA0
5  // U0Tx (VCP transmit) connected to PA1
6
7  #include "UART0.h"
8  #include "tm4c123gh6pm.h"
9  #include "util.h"
10
11 //------------UART_Init------------
12 // Initialize the UART for 57600 baud rate (assuming 16 MHz UART clock),
13 // 8 bit word length, no parity bits, one stop bit, FIFOs enabled
14 // Input: none
15 // Output: none
16 void UART0_Init(void){
17   SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART0; // activate UART0
18   SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
19   UART0_CTL_R = 0;                      // disable UART
```

```
  UART0_IBRD_R = 17;            // IBRD = ?
  UART0_FBRD_R = 23;            // FBRD = ?
                                // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
  UART0_CTL_R |= UART_CTL_RXE|UART_CTL_TXE|UART_CTL_UARTEN;// enable Tx, RX and UART
  GPIO_PORTA_AFSEL_R |= 0x03;           // enable alt funct on PA1-0
  GPIO_PORTA_DEN_R |= 0x03;             // enable digital I/O on PA1-0
                                        // configure PA1-0 as UART
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTA_AMSEL_R &= ~0x03;          // disable analog functionality on PA
}


//---------------------OutCRLF---------------------
// Output a CR,LF to UART to go to a new line
// Input: none
// Output: none

void UART0_OutCRLF(void){
  UART0_OutChar(CR);
  UART0_OutChar(LF);
}

//------------UART_InChar------------
// Wait for new serial port input
// Input: none
// Output: ASCII code for key typed
unsigned char UART0_InChar(void){
  while((UART0_FR_R&UART_FR_RXFE) != 0); // wait until the receiving FIFO is not empty
  return((unsigned char)(UART0_DR_R&0xFF));
```

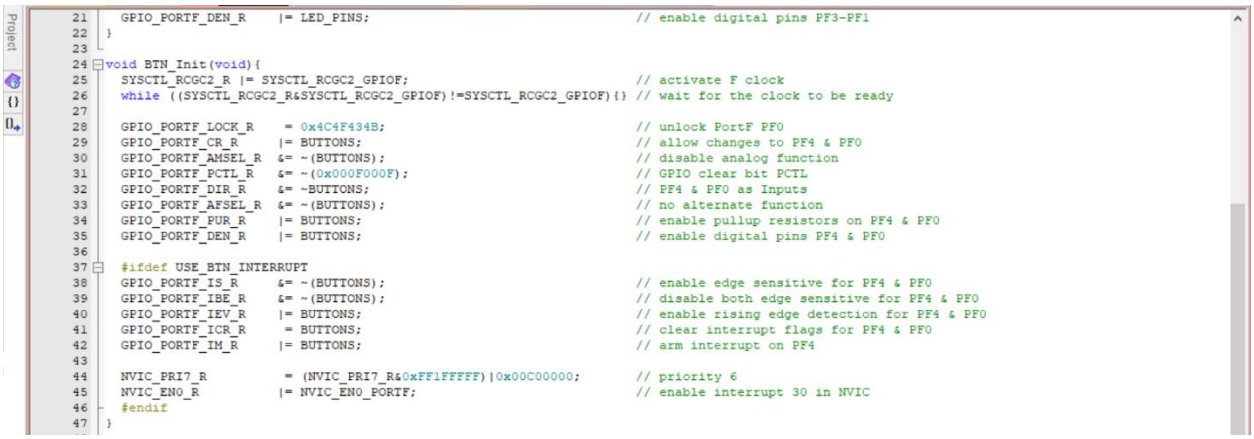
```
}
//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART0_OutChar(char data){
  while((UART0_FR_R&UART_FR_TXFF) != 0);
  UART0_DR_R = data;
}


//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART0_OutString(char *pt){
  while(*pt){
    UART0_OutChar(*pt);
    pt++;
  }
  //UART0_OutChar(0); // add the null terminator
}

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//    and adds them to a string until <enter> is typed
//    or until max length of the string is reached.
//    when max length is reach, no more input will be accepted
//    the display will wait for the <enter> key to be pressed.
// It echoes each character as it is inputted.
```

```
// If a backspace is inputted, the string is modified
//    and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
// -- Modified by Agustinus Darmawan + Mingjie Qiu --
void UART0_InString(char *bufPt, unsigned short max) {
int length=0;
char character;
  character = UART0_InChar();
  while(character != CR){
    if(character == BS){ // back space
      if(length){
        bufPt--;
        length--;
        UART0_OutChar(BS);
      }
    }
    else if(length < max){
      *bufPt = character;
      bufPt++;
      length++;
      UART0_OutChar(character);
    }
    character = UART0_InChar();
  }
  *bufPt = 0; // adding null terminator to the end of the string.
}
```

ButtonLED.c

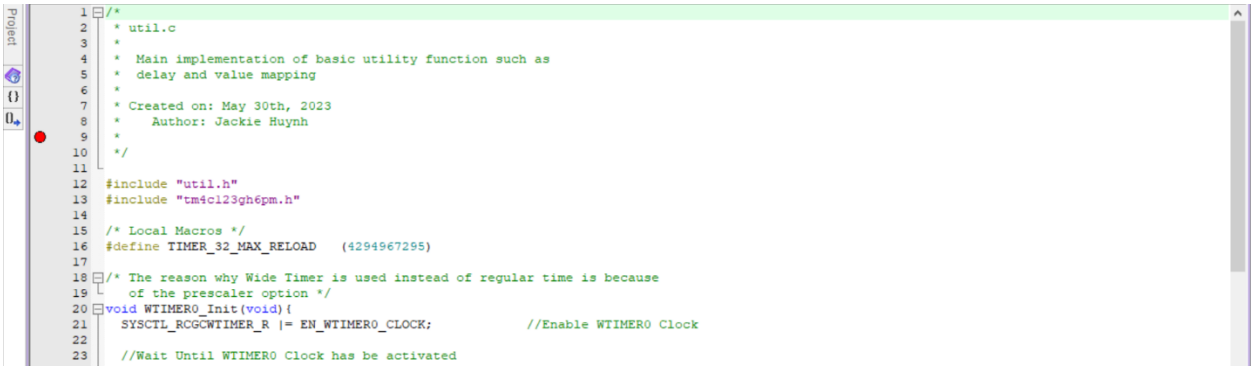


```c
/*
 * ButtonLED.c
 *
 * Created on: March 1st, 2023
 *     Author: Jackie Huynh
 *
 */

#include "ButtonLED.h"

void LED_Init(void){
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF;                          // activate F clock
  while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOF)!=SYSCTL_RCGC2_GPIOF){} // wait for the clock to be ready

  GPIO_PORTF_CR_R     |= LED_PINS;                               // allow changes to PF3-1
  GPIO_PORTF_AMSEL_R  &= ~(LED_PINS);                            // disable analog function
  GPIO_PORTF_PCTL_R   &= ~(0x0000FFF0);                          // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R    |= LED_PINS;                               // PF3, PF2, PF1 LED Output
  GPIO_PORTF_AFSEL_R  &= ~(LED_PINS);                            // no alternate function
  GPIO_PORTF_PUR_R    &= ~(LED_PINS);                            // disable pullup resistors on PF3, PF2, PF1
```



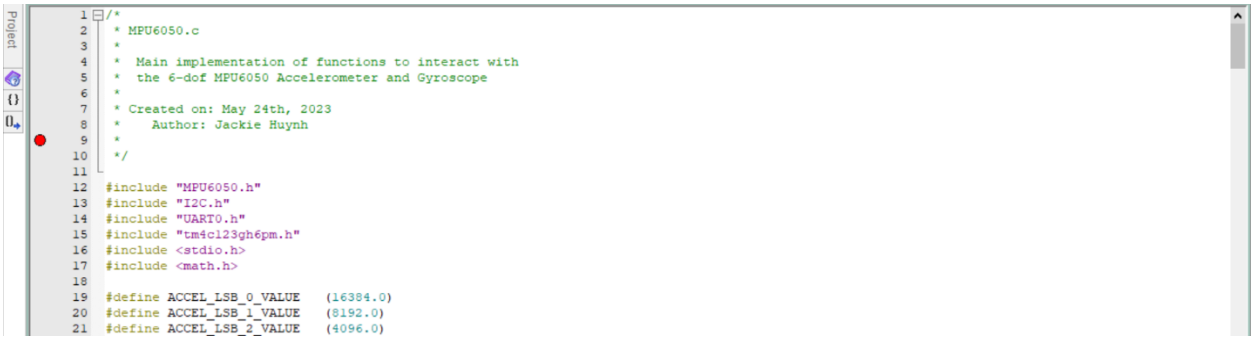
```c
  GPIO_PORTF_DEN_R    |= LED_PINS;                               // enable digital pins PF3-PF1
}

void BTN_Init(void){
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF;                          // activate F clock
  while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOF)!=SYSCTL_RCGC2_GPIOF){} // wait for the clock to be ready

  GPIO_PORTF_LOCK_R    = 0x4C4F434B;                             // unlock PortF PF0
  GPIO_PORTF_CR_R     |= BUTTONS;                                // allow changes to PF4 & PF0
  GPIO_PORTF_AMSEL_R  &= ~(BUTTONS);                             // disable analog function
  GPIO_PORTF_PCTL_R   &= ~(0x000F000F);                          // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R    &= ~BUTTONS;                               // PF4 & PF0 as Inputs
  GPIO_PORTF_AFSEL_R  &= ~(BUTTONS);                             // no alternate function
  GPIO_PORTF_PUR_R    |= BUTTONS;                                // enable pullup resistors on PF4 & PF0
  GPIO_PORTF_DEN_R    |= BUTTONS;                                // enable digital pins PF4 & PF0

  #ifdef USE_BTN_INTERRUPT
  GPIO_PORTF_IS_R     &= ~(BUTTONS);                             // enable edge sensitive for PF4 & PF0
  GPIO_PORTF_IBE_R    &= ~(BUTTONS);                             // disable both edge sensitive for PF4 & PF0
  GPIO_PORTF_IEV_R    |= BUTTONS;                                // enable rising edge detection for PF4 & PF0
  GPIO_PORTF_ICR_R     = BUTTONS;                                // clear interrupt flags for PF4 & PF0
  GPIO_PORTF_IM_R     |= BUTTONS;                                // arm interrupt on PF4

  NVIC_PRI7_R          = (NVIC_PRI7_R&0xFF1FFFFF)|0x00C00000;    // priority 6
  NVIC_EN0_R          |= NVIC_EN0_PORTF;                         // enable interrupt 30 in NVIC
  #endif
}
```

Util.c



```c
/*
 * util.c
 *
 *  Main implementation of basic utility function such as
 *  delay and value mapping
 *
 * Created on: May 30th, 2023
 *     Author: Jackie Huynh
 *
 */

#include "util.h"
#include "tm4c123gh6pm.h"

/* Local Macros */
#define TIMER_32_MAX_RELOAD   (4294967295)

/* The reason why Wide Timer is used instead of regular time is because
   of the prescaler option */
void WTIMER0_Init(void){
  SYSCTL_RCGCWTIMER_R |= EN_WTIMER0_CLOCK;           //Enable WTIMER0 Clock

  //Wait Until WTIMER0 Clock has be activated
  while((SYSCTL_RCGCWTIMER_R&EN_WTIMER0_CLOCK)!=EN_WTIMER0_CLOCK);

  WTIMER0_CTL_R &= ~(WTIMER0_TAEN_BIT);               //Disable WTIMER0 Timer A
  WTIMER0_CFG_R |= WTIMER0_32_BIT_CFG;               //Set WTIMER0 to be 32-bit config mode
  WTIMER0_TAMR_R |= WTIMER0_PERIOD_MODE;             //Set WTIMER0 to be in periodic mode

  /* Prescale down to 1MHz or 1us period */
  // Frequency = System Clock / Prescaler
  // Frequency = 40MHz / 40000 = 0.001MHz = 1kHz
  // Tick Length = Period = 1 / 1kHz = 1ms
  WTIMER0_TAPR_R = PRESCALER_VALUE;                  //Set prescaler to get 1kHz frequency or 1ms period
}

void DELAY_1MS(uint32_t delay){
  WTIMER0_TAILR_R = delay - 1;
  WTIMER0_CTL_R |= WTIMER0_TAEN_BIT;
  while(WTIMER0_TAR_R != 0);
  WTIMER0_CTL_R &= ~(WTIMER0_TAEN_BIT);
}

int16_t map(int16_t x, int16_t x_min, int16_t x_max, int16_t out_min, int16_t out_max){
  if(x < x_min){
    return x_min;
  }
  if(x > x_max){
    return x_max;
  }
  return (x - x_min) * (out_max - out_min) / (x_max - x_min) + out_min;
}
```

MPU6050.c



```c
/*
 * MPU6050.c
 *
 *  Main implementation of functions to interact with
 *  the 6-dof MPU6050 Accelerometer and Gyroscope
 *
 * Created on: May 24th, 2023
 *     Author: Jackie Huynh
 *
 */

#include "MPU6050.h"
#include "I2C.h"
#include "UART0.h"
#include "tm4c123gh6pm.h"
#include <stdio.h>
#include <math.h>

#define ACCEL_LSB_0_VALUE    (16384.0)
#define ACCEL_LSB_1_VALUE    (8192.0)
#define ACCEL_LSB_2_VALUE    (4096.0)
```

```
#define ACCEL_LSB_3_VALUE    (2048.0)

#define GYRO_LSB_0_VALUE     (131.0)
#define GYRO_LSB_1_VALUE     (65.5)
#define GYRO_LSB_2_VALUE     (32.8)
#define GYRO_LSB_3_VALUE     (16.4)

/*
 *  --------------------MPU6050_Init---------------------
 *  Basic Initialization Function for MPU6050 @ default settings
 *  Input: none
 *  Output: none
 */
void MPU6050_Init(void){

  uint8_t ret;
  char stringBuf[10];

  //If check does not equal to their respected address, MPU is not detected
  #ifndef USE_HIGH
  ret = I2C0_Receive(MPU6050_ADDR_AD0_LOW, WHO_AM_I);
  if(ret != MPU6050_ADDR_AD0_LOW){
    UART0_OutString("MPU6050 has not been Detected\r\n");
    return;
  }
  #else
  ret = I2C0_Receive(MPU6050_ADDR_AD0_HIGH, WHO_AM_I);
  if(ret != MPU6050_ADDR_AD0_HIGH){
    UART0_OutString("MPU6050 has not been Detected\r\n");
    return;
```

```
  }
  #endif

  //Print ID out to terminal
  sprintf(stringBuf, "ID: %x\r\n", ret);
  UART0_OutString(stringBuf);

  UART0_OutString("MPU6050 has been Detected\r\n");
  UART0_OutString("MPU6050 is initializing\r\n");

  /* Reset the MPU6050 Module */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, PWR_MGMT_1, PWR_DEVICE_RESET);
  UART0_OutString("Reset MPU6050\r\n");

  /* 0 to wake up sensor */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, PWR_MGMT_1, PWR_CLK_SEL_INTERNAL);
  if(ret != 0)
    UART0_OutString("Error On Transmit\r\n");
  else
    UART0_OutString("Sensor is awake\r\n");

  /* Set Data Rate to 1kHz */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, SMPLRT_DIV, SMPLRT_DIV_8);
  if(ret != 0)
    UART0_OutString("Error On Transmit\r\n");
  else
    UART0_OutString("Data Rate is 1kHz\r\n");

  /* Default Configuration */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, CONFIG, CONFIG_DFPL_0);
```

```
  if(ret != 0)
    UART0_OutString("Error On Transmit\r\n");
  else
    UART0_OutString("Default Configuration\r\n");

  /* Default config for Accelerometer */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, ACCEL_CONFIG, ACCEL_AFS_SEL_0);
  if(ret != 0)
    UART0_OutString("Error On Transmit\r\n");
  else
    UART0_OutString("Default Accelerometer Configuration\r\n");

  /* Default config for Gyroscope */
  ret = I2C0_Transmit(MPU6050_ADDR_AD0_LOW, GYRO_CONFIG, GYRO_FS_SEL_0);
  if(ret != 0)
    UART0_OutString("Error On Transmit\r\n");
  else
    UART0_OutString("Default Gyroscope Configuration\r\n");

  UART0_OutString("MPU6050 Initialized\r\n");
}

/*
 *  -----------------MPU6050_Get_Accel------------------
 *  Receive Raw Accelerometer Data and store it in the user struct
 *  Input: MPU6050 Accel User Instance Struct
 *  Output: none
 */
void MPU6050_Get_Accel(MPU6050_ACCEL_t* Accel_Instance){

```

```
112   /* Local Variables */
113   uint8_t ACCEL_X_LOW;
114   uint8_t ACCEL_X_HIGH;
115   uint8_t ACCEL_Y_LOW;
116   uint8_t ACCEL_Y_HIGH;
117   uint8_t ACCEL_Z_LOW;
118   uint8_t ACCEL_Z_HIGH;
119
120   /* Grab 16-bit Accel data of each axis by reading ACCEL data register using I2C*/
121   ACCEL_X_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_XOUT_L);
122   ACCEL_X_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_XOUT_H);
123   ACCEL_Y_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_YOUT_L);
124   ACCEL_Y_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_YOUT_H);
125   ACCEL_Z_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_ZOUT_L);
126   ACCEL_Z_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_ZOUT_H);
127
128   /* Concatanate and Save Into Accelerometer Struct Instance */
129   Accel_Instance->Ax_RAW = (ACCEL_X_HIGH << 8) | ACCEL_X_LOW;
130   Accel_Instance->Ay_RAW = (ACCEL_Y_HIGH << 8) | ACCEL_Y_LOW;
131   Accel_Instance->Az_RAW = (ACCEL_Z_HIGH << 8) | ACCEL_Z_LOW;
132
133 }
134
135 /*
136  *  ------------------MPU6050_Get_Gyro-------------------
137  *  Receive Raw Gyroscope Data and store it in the user struct
138  *  Input: MPU6050 Gyro User Instance Struct
139  *  Output: none
140  */
141 void MPU6050_Get_Gyro(MPU6050_GYRO_t* Gyro_Instance){
```

```
142
143   /* Local Variables */
144   uint8_t GYRO_X_LOW;
145   uint8_t GYRO_X_HIGH;
146   uint8_t GYRO_Y_LOW;
147   uint8_t GYRO_Y_HIGH;
148   uint8_t GYRO_Z_LOW;
149   uint8_t GYRO_Z_HIGH;
150
151   /* Grab 16-bit Gyro Data of each Axis by reading GYRO data register using I2C*/
152   GYRO_X_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_XOUT_L);
153   GYRO_X_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_XOUT_H);
154   GYRO_Y_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_YOUT_L);
155   GYRO_Y_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_YOUT_H);
156   GYRO_Z_LOW = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_ZOUT_L);
157   GYRO_Z_HIGH = I2C0_Receive(MPU6050_ADDR_AD0_LOW, GYRO_ZOUT_H);
158
159   /* Concatanate and Save Into Gyro Struct Instance */
160   Gyro_Instance->Gx_RAW = (GYRO_X_HIGH << 8) | GYRO_X_LOW;
161   Gyro_Instance->Gy_RAW = (GYRO_Y_HIGH << 8) | GYRO_Y_LOW;
162   Gyro_Instance->Gz_RAW = (GYRO_Z_HIGH << 8) | GYRO_Z_LOW;
163 }
164
165 /*
166  *  ---------------MPU6050_Process_Accel----------------
167  *  Process Raw Accelerometer Data into usable data and store
168  *  it in the user stuct
169  *  Input: MPU6050 Accel User Instance Struct
170  *  Output: none
171  */
```

```
172 void MPU6050_Process_Accel(MPU6050_ACCEL_t* Accel_Instance){
173
174   char LSB_Sensitivity;
175
176   //Read LSB Sensitivity Setting from ACCEL_CONFIG Register
177   #ifndef USE_HIGH
178     LSB_Sensitivity = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_CONFIG);
179   #else
180     LSB_Sensitivity = I2C0_Receive(MPU6050_ADDR_AD0_HIGH, ACCEL_CONFIG);
181   #endif
182
183   //Based on setting, process raw data accordingly
184   switch(LSB_Sensitivity){
185     case ACCEL_AFS_SEL_0:
186       Accel_Instance->Ax = (float)Accel_Instance->Ax_RAW / ACCEL_LSB_0_VALUE;
187       Accel_Instance->Ay = (float)Accel_Instance->Ay_RAW / ACCEL_LSB_0_VALUE;
188       Accel_Instance->Az = (float)Accel_Instance->Az_RAW / ACCEL_LSB_0_VALUE;
189       break;
190     case ACCEL_AFS_SEL_1:
191       Accel_Instance->Ax = (float)Accel_Instance->Ax_RAW / ACCEL_LSB_1_VALUE;
192       Accel_Instance->Ay = (float)Accel_Instance->Ay_RAW / ACCEL_LSB_1_VALUE;
193       Accel_Instance->Az = (float)Accel_Instance->Az_RAW / ACCEL_LSB_1_VALUE;
194       break;
195     case ACCEL_AFS_SEL_2:
196       Accel_Instance->Ax = (float)Accel_Instance->Ax_RAW / ACCEL_LSB_2_VALUE;
197       Accel_Instance->Ay = (float)Accel_Instance->Ay_RAW / ACCEL_LSB_2_VALUE;
198       Accel_Instance->Az = (float)Accel_Instance->Az_RAW / ACCEL_LSB_2_VALUE;
199       break;
200     case ACCEL_AFS_SEL_3:
201       Accel_Instance->Ax = (float)Accel_Instance->Ax_RAW / ACCEL_LSB_3_VALUE;
```

```
202         Accel_Instance->Ay = (float)Accel_Instance->Ay_RAW / ACCEL_LSB_3_VALUE;
203         Accel_Instance->Az = (float)Accel_Instance->Az_RAW / ACCEL_LSB_3_VALUE;
204         break;
205     }
206 }
207
208 /*
209  *  ---------------MPU6050_Process_Gyro----------------
210  *  Process Raw Gyroscope Data into usable data and store it in
211  *  the user struct
212  *  Input: MPU6050 Gyro User Instance Struct
213  *  Output: none
214  */
215 void MPU6050_Process_Gyro(MPU6050_GYRO_t* Gyro_Instance){
216
217   char LSB_Sensitivity;
218
219   //Read LSB Sensitivity Setting from GYRO_CONFIG Register
220   #ifndef USE_HIGH
221     LSB_Sensitivity = I2C0_Receive(MPU6050_ADDR_AD0_LOW, ACCEL_CONFIG);
222   #else
223     LSB_Sensitivity = I2C0_Receive(MPU6050_ADDR_AD0_HIGH, ACCEL_CONFIG);
224   #endif
225
226   //Based on setting, process raw data accordingly
227   switch(LSB_Sensitivity){
228     case GYRO_FS_SEL_0:
229       Gyro_Instance->Gx = (float)Gyro_Instance->Gx_RAW / GYRO_LSB_0_VALUE;
230       Gyro_Instance->Gy = (float)Gyro_Instance->Gy_RAW / GYRO_LSB_0_VALUE;
231       Gyro_Instance->Gz = (float)Gyro_Instance->Gz_RAW / GYRO_LSB_0_VALUE;
```

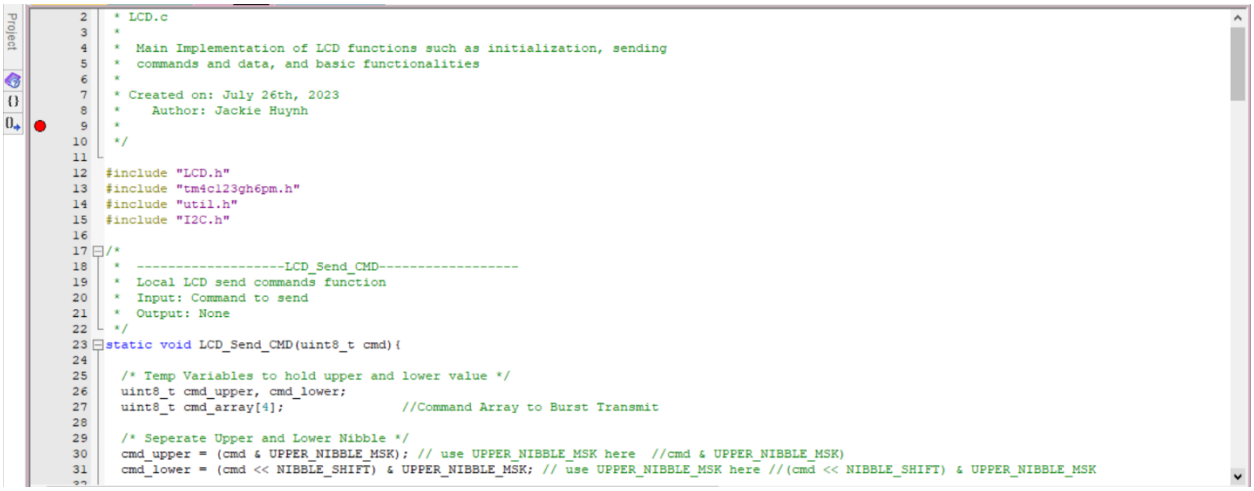
```
232       break;
233     case GYRO_FS_SEL_1:
234       Gyro_Instance->Gx = (float)Gyro_Instance->Gx_RAW / GYRO_LSB_1_VALUE;
235       Gyro_Instance->Gy = (float)Gyro_Instance->Gy_RAW / GYRO_LSB_1_VALUE;
236       Gyro_Instance->Gz = (float)Gyro_Instance->Gz_RAW / GYRO_LSB_1_VALUE;
237       break;
238     case GYRO_FS_SEL_2:
239       Gyro_Instance->Gx = (float)Gyro_Instance->Gx_RAW / GYRO_LSB_2_VALUE;
240       Gyro_Instance->Gy = (float)Gyro_Instance->Gy_RAW / GYRO_LSB_2_VALUE;
241       Gyro_Instance->Gz = (float)Gyro_Instance->Gz_RAW / GYRO_LSB_2_VALUE;
242       break;
243     case GYRO_FS_SEL_3:
244       Gyro_Instance->Gx = (float)Gyro_Instance->Gx_RAW / GYRO_LSB_3_VALUE;
245       Gyro_Instance->Gy = (float)Gyro_Instance->Gy_RAW / GYRO_LSB_3_VALUE;
246       Gyro_Instance->Gz = (float)Gyro_Instance->Gz_RAW / GYRO_LSB_3_VALUE;
247       break;
248   }
249 }
250
251 /*
252  *  ----------------MPU6050_Get_Angle----------------
253  *  Calculate Tilt Angle using processed Accelerometer and
254  *  Gyroscope data and it in the user angle struct
255  *  Input: MPU6050 Angle User Instance Struct
256  *  Output: none
257  */
258 void MPU6050_Get_Angle(MPU6050_ACCEL_t* Accel_Instance, MPU6050_GYRO_t* Gyro_Instance, MPU6050_ANGLE_t* Angle_Instance){
259
260   //Hint: Use RAD_TO_DEGREE_CONV macro to convert radian to degree
261   /*CODE_FILL*/
```

```
262     float Accel_Angle_X, Accel_Angle_Y;
263
264     // Calculate Accelerometer angles
265     Accel_Angle_X = atan2(Accel_Instance->Ay, sqrt(pow(Accel_Instance->Ax, 2) + pow(Accel_Instance->Az, 2)));
266     Accel_Angle_Y = atan2(-Accel_Instance->Ax, sqrt(pow(Accel_Instance->Ay, 2) + pow(Accel_Instance->Az, 2)));
267
268     // Convert to degrees
269     Accel_Angle_X *= RAD_TO_DEGREE_CONV;
270     Accel_Angle_Y *= RAD_TO_DEGREE_CONV;
271
272     // Gyroscope angles are already in degrees
273     Angle_Instance->ArX = Accel_Angle_X;
274     Angle_Instance->ArY = Accel_Angle_Y;
275
276     // Gyroscope Z angle is just cumulative angle from gyroscope data
277     Angle_Instance->ArZ += Gyro_Instance->Gz / 131.0; // 131.0 is the sensitivity scale factor for the gyroscope
278
279     // Ensure angle stays within -180 to 180 degrees
280     if (Angle_Instance->ArZ > 180) {
281         Angle_Instance->ArZ -= 360;
282     } else if (Angle_Instance->ArZ < -180) {
283         Angle_Instance->ArZ += 360;
284     }
285
286 }
287
288 /* Used for Debugging Purposes */
289 uint8_t MPU6050_Read_Reg(uint8_t reg){
290   return I2C0_Receive(MPU6050_ADDR_AD0_LOW, reg);
291 }
```

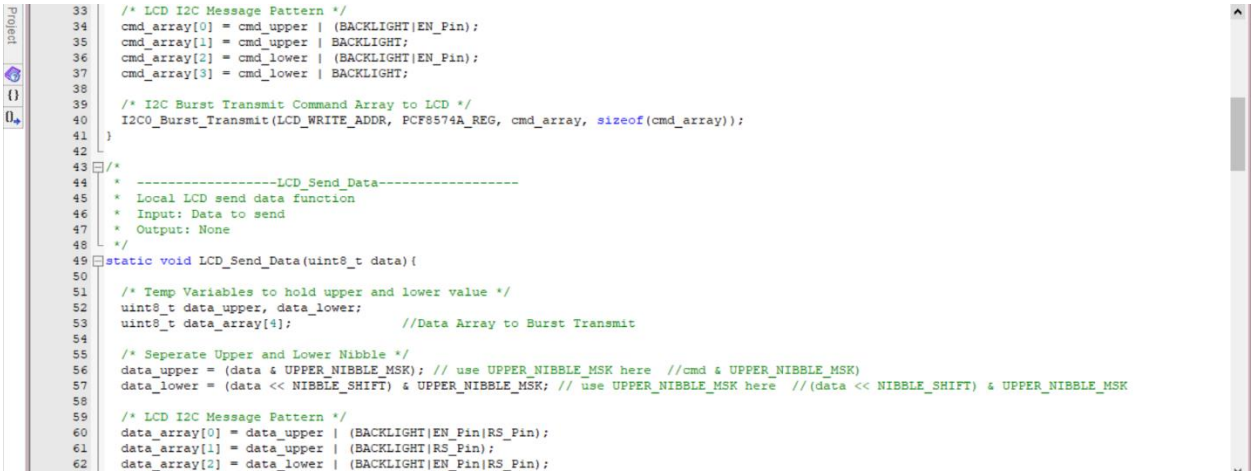
Servo.c

```c
/*
 * Servo.c
 *
 *  Implementations of functions to interact with a standard Servo Motor
 *
 * Created on: June 11th, 2023
 *    Author: Jackie Huynh
 *
 */

#include "Servo.h"
#include "util.h"
#include "tm4c123gh6pm.h"

/*
 *  --------------------Servo_Init------------------
 *  Basic Servo Initialization function for PWM Generation
 *  Input: None
 *  Output: None
 */
void Servo_Init(void){
  SYSCTL_RCGC2_R |= EN_PWM0_GPIOB_CLOCK;        //Activate GPIOB Clock
  while((SYSCTL_RCGC2_R&EN_PWM0_GPIOB_CLOCK)!=EN_PWM0_GPIOB_CLOCK);

  /* GPIO Configuration */
  GPIO_PORTB_AFSEL_R  |= PWM0_PIN;              //Enable Alternate Function on PB6
  GPIO_PORTB_PCTL_R   &= ~CLEAR_ALT_FUNCTION;   //Clear any Previous Alternate Function
  GPIO_PORTB_PCTL_R   |= PWM0_ALT_FUNCTION;     //Set Alternate Function to PWM on PB6
  GPIO_PORTB_AMSEL_R  &= ~(PWM0_PIN);           //Disable Analog Function
  GPIO_PORTB_DEN_R    |= PWM0_PIN;              //Enable Digital I/O on PB6
  GPIO_PORTB_DR8R_R   |= PWM0_PIN;              //Enable 8mA Drive

  /* PWM Clock Configuration */
  SYSCTL_RCGCPWM_R |= EN_PWM0_CLOCK;            //Activate PWM0 Module

  /*
  Servo Signal Requires 50Hz or 20ms Period
  Default 16MHz and a 16-bit counter won't be enough
  Will Need to Prescale the PWM Clock

  Example:
  PWM Clock = System Clock / Prescaler = 40MHz / 16 = 2.5MHz
  Time Per Tick = 1 / PWM Clock = 1 / 2.5MHz = 0.4us
  Period = Counter * Time Per Tick
  Counter = Period / Time Per Tick = 20ms / 0.4us = 50000 - 1 = 49999
  Frequency = 1 / Period = 1 / 20ms = 50Hz
  */
  SYSCTL_RCC_R |= EN_USE_PWM_DIV;       //Enable PWM Divider
  SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M;       //Clear PWM divider field
  SYSCTL_RCC_R |= PWM0_DIV_VALUE;       //Set PWM Divider to 8

  /* PWM Gen 0 Configuration */
  PWM0_0_CTL_R = PWM0_DEFAULT_CONFIG;   //Default mode of count-down and auto-reload

  PWM0_0_GENA_R |= PWM0_GEN_CONFIG;     //low on LOAD, high on CMPA match

  PWM0_0_LOAD_R  = PWM0_PERIOD-1;     //cycles needed to count down to 0
  PWM0_0_CMPA_R  = 0;                   //Set to 0% duty cycle
  PWM0_0_CTL_R  |= PWM0_START;          //Start PWM0 Gen 0
  PWM0_ENABLE_R |= EN_PWM0_FUNCTION;    //Enable PWM0

}
/*
 *  --------------------Drive_Servo------------------
 *  Drives the Servo Motor to a specified angle
 *  Input: Desired Angle
 *  Output: None
 */
void Drive_Servo(int16_t angle){

  /* Temp Variable to Hold value that is being mapped*/
  int16_t mapped;

  /* Assert Parameter */
  if((angle < SERVO_MIN_ANGLE) || (angle > SERVO_MAX_ANGLE))
    return;

  /* Map Function */
  mapped = map(angle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE, SERVO_MIN_CNT-1, SERVO_MAX_CNT-1);

  /* Setting New Compare Value */
  PWM0_0_CMPA_R = mapped;

}
```

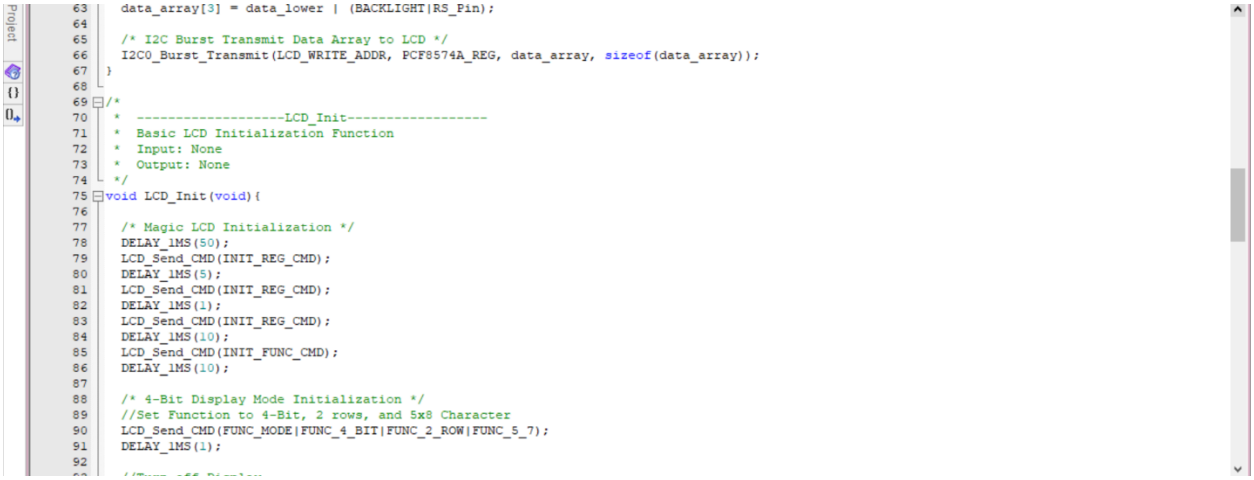
LCD.c



```
 * LCD.c
 *
 *  Main Implementation of LCD functions such as initialization, sending
 *  commands and data, and basic functionalities
 *
 * Created on: July 26th, 2023
 *     Author: Jackie Huynh
 *
 */

#include "LCD.h"
#include "tm4c123gh6pm.h"
#include "util.h"
#include "I2C.h"

/*
 *  ------------------LCD_Send_CMD------------------
 *  Local LCD send commands function
 *  Input: Command to send
 *  Output: None
 */
static void LCD_Send_CMD(uint8_t cmd){

  /* Temp Variables to hold upper and lower value */
  uint8_t cmd_upper, cmd_lower;
  uint8_t cmd_array[4];              //Command Array to Burst Transmit

  /* Seperate Upper and Lower Nibble */
  cmd_upper = (cmd & UPPER_NIBBLE_MSK); // use UPPER_NIBBLE_MSK here  //cmd & UPPER_NIBBLE_MSK)
  cmd_lower = (cmd << NIBBLE_SHIFT) & UPPER_NIBBLE_MSK; // use UPPER_NIBBLE_MSK here //(cmd << NIBBLE_SHIFT) & UPPER_NIBBLE_MSK
```



```
  /* LCD I2C Message Pattern */
  cmd_array[0] = cmd_upper | (BACKLIGHT|EN_Pin);
  cmd_array[1] = cmd_upper | BACKLIGHT;
  cmd_array[2] = cmd_lower | (BACKLIGHT|EN_Pin);
  cmd_array[3] = cmd_lower | BACKLIGHT;

  /* I2C Burst Transmit Command Array to LCD */
  I2C0_Burst_Transmit(LCD_WRITE_ADDR, PCF8574A_REG, cmd_array, sizeof(cmd_array));
}

/*
 *  ------------------LCD_Send_Data------------------
 *  Local LCD send data function
 *  Input: Data to send
 *  Output: None
 */
static void LCD_Send_Data(uint8_t data){

  /* Temp Variables to hold upper and lower value */
  uint8_t data_upper, data_lower;
  uint8_t data_array[4];              //Data Array to Burst Transmit

  /* Seperate Upper and Lower Nibble */
  data_upper = (data & UPPER_NIBBLE_MSK); // use UPPER_NIBBLE_MSK here  //cmd & UPPER_NIBBLE_MSK)
  data_lower = (data << NIBBLE_SHIFT) & UPPER_NIBBLE_MSK; // use UPPER_NIBBLE_MSK here  //(data << NIBBLE_SHIFT) & UPPER_NIBBLE_MSK

  /* LCD I2C Message Pattern */
  data_array[0] = data_upper | (BACKLIGHT|EN_Pin|RS_Pin);
  data_array[1] = data_upper | (BACKLIGHT|RS_Pin);
  data_array[2] = data_lower | (BACKLIGHT|EN_Pin|RS_Pin);
```



```
  data_array[3] = data_lower | (BACKLIGHT|RS_Pin);

  /* I2C Burst Transmit Data Array to LCD */
  I2C0_Burst_Transmit(LCD_WRITE_ADDR, PCF8574A_REG, data_array, sizeof(data_array));
}

/*
 *  ------------------LCD_Init------------------
 *  Basic LCD Initialization Function
 *  Input: None
 *  Output: None
 */
void LCD_Init(void){

  /* Magic LCD Initialization */
  DELAY_1MS(50);
  LCD_Send_CMD(INIT_REG_CMD);
  DELAY_1MS(5);
  LCD_Send_CMD(INIT_REG_CMD);
  DELAY_1MS(1);
  LCD_Send_CMD(INIT_REG_CMD);
  DELAY_1MS(10);
  LCD_Send_CMD(INIT_FUNC_CMD);
  DELAY_1MS(10);

  /* 4-Bit Display Mode Initialization */
  //Set Function to 4-Bit, 2 rows, and 5x8 Character
  LCD_Send_CMD(FUNC_MODE|FUNC_4_BIT|FUNC_2_ROW|FUNC_5_7);
  DELAY_1MS(1);
```

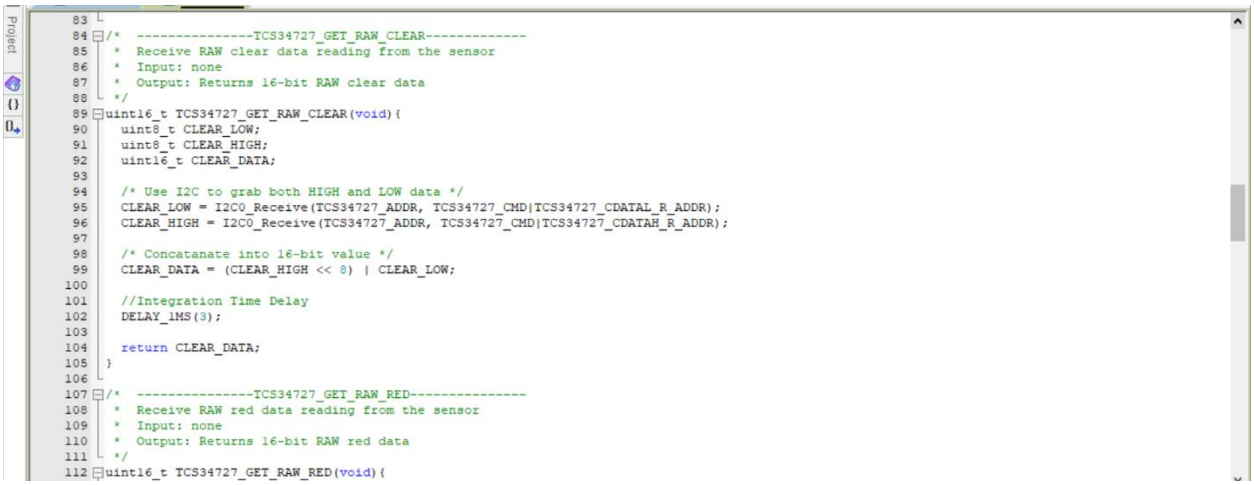
```
93    //Turn off Display
94    LCD_Send_CMD(DISP_CMD|DISP_OFF|DISP_CURSOR_OFF|DISP_BLINK_OFF);
95    DELAY_1MS(1);
96
97    //Clear Display
98    LCD_Send_CMD(CLEAR_DISP_CMD);
99    DELAY_1MS(2);
100
101   //Set Entry Mode
102   LCD_Send_CMD(ENTRY_MODE_CMD|ENTRY_INC_CURSOR);
103   DELAY_1MS(1);
104
105   //Turn on Display with cursor and blink enable
106   LCD_Send_CMD(DISP_CMD|DISP_ON|DISP_CURSOR_ON|DISP_BLINK_ON);
107
108 }
109
110 /*
111  *  -------------------LCD_Clear------------------
112  *  Clear the LCD Display by passing a command
113  *  Input: None
114  *  Output: None
115  */
116 void LCD_Clear(void){
117   LCD_Send_CMD(CLEAR_DISP_CMD);
118   DELAY_1MS(1);
119 }
120
121 /*
122  *  ----------------LCD_Set_Cursor----------------
```

```
123  *  Set Cursor to Desire Place
124  *  Input: Desired Row and Column to place Cursor
125  *  Output: None
126  */
127 void LCD_Set_Cursor(uint8_t row, uint8_t col){
128
129   /* Switch case based of row choice */
130   switch(row){
131     case ROW1:
132       col |= FIRST_ROW_CMD;
133       break;
134     case ROW2:
135       col |= SECOND_ROW_CMD;
136       break;
137     default:
138       col |= FIRST_ROW_CMD;
139       break;
140   }
141
142   /* Send Command to set Row and Column */
143   LCD_Send_CMD(col);
144   DELAY_1MS(2);
145
146 }
147
148 /*
149  *  ---------------LCD_Reset_Cursor---------------
150  *  Reset Cursor back to Row 1 and Column 0
151  *  Input: None
152  *  Output: None
```

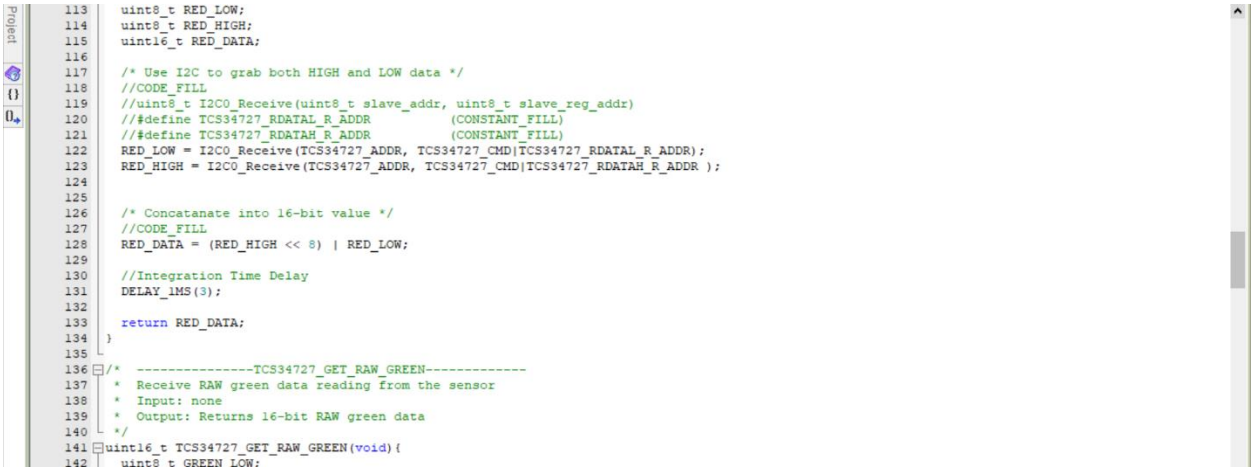
```
153  */
154 void LCD_Reset_Cursor(void){
155   LCD_Send_CMD(RETURN_HOME_CMD);
156   DELAY_1MS(1);
157 }
158
159 /*
160  *  ----------------LCD_Print_Char----------------
161  *  Prints a Character to LCD
162  *  Input: Character Hex Value
163  *  Output: None
164  */
165 void LCD_Print_Char(uint8_t data){
166   LCD_Send_Data(data);
167   DELAY_1MS(1);
168 }
169
170 /*
171  *  ----------------LCD_Print_Str----------------
172  *  Prints a string to LCD
173  *  Input: Pointer to Character Array
174  *  Output: None
175  */
176 void LCD_Print_Str(uint8_t* str){
177   while(*str){
178     LCD_Send_Data(*str++);
179     DELAY_1MS(2);
180   }
181 }
```

TCS34727.c

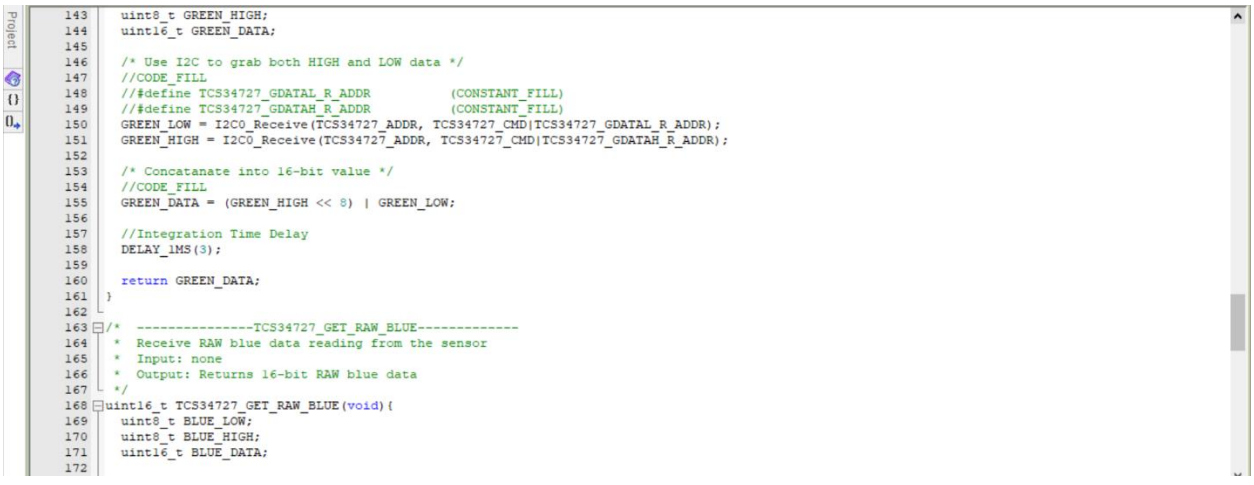
```c
/*
 * TCS34727.c
 *
 *  Main implementation of the functions to interact with
 *  the TCS34727 RGB Color Sensor
 *
 * Created on: May 24th, 2023
 *     Author: Jackie Huynh
 *
 */

#include "TCS34727.h"
#include "I2C.h"
#include "UART0.h"
#include "util.h"
#include <stdio.h>
#include "tm4c123gh6pm.h"

/*  ------------------TCS34727_Init-----------------
 *  Basic Initialization Function for TCS34727 at default settings
 *  Input: none
 *  Output: none
 */
void TCS34727_Init(void){
  uint8_t ret;                          //Temp Variable to hold return values
  char printBuf[20];                    //String buffer to print

  /* Check if RGB Color Sensor has been detected */
  ret = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_ID_R_ADDR);

  //Print ID or Error to Terminal
  sprintf(printBuf, "ID: %x\r\n", ret);
  UART0_OutString(printBuf);

  if(ret != TCS34727_ID){
    UART0_OutString("TCS34727 has not been Detected\r\n");
    return;
  }
  UART0_OutString("TCS34727 has been Detected\r\n");

  /* Set Integration Time to 2.4ms in timing register */
  ret = I2C0_Transmit(TCS34727_ADDR, TCS34727_CMD|TCS34727_TIMING_R_ADDR, TCS34727_ATIME_2_4_MS);
  if(ret != 0)
    UART0_OutString("Error on Transmit\r\n");
  else
    UART0_OutString("TCS34727 Integration Time Set\r\n");

  // Necessary Delay when setting integration time/wait time.
  // This varies for which integration time is choosen.
  // This project chooses 2.4ms.
  DELAY_1MS(3);

  /* Setting Gain to 1X gain */
  ret = I2C0_Transmit(TCS34727_ADDR, TCS34727_CMD|TCS34727_CTRL_R_ADDR, TCS34727_CTRL_AGAIN_1);
  if(ret != 0)
    UART0_OutString("Error on Transmit\r\n");
  else
    UART0_OutString("TCS34727 Gain Set\r\n");

  /* Powering On Sensor at Enable register */
  ret = I2C0_Transmit(TCS34727_ADDR, TCS34727_CMD|TCS34727_ENABLE_R_ADDR, TCS34727_ENABLE_PON);
  if(ret != 0)
    UART0_OutString("Error on Transmit\r\n");
  else
    UART0_OutString("TCS34727 Power On\r\n");

  //Nessessary Delay When Powering On Module
  DELAY_1MS(3);

  /* Enabling RGBC 2-Channel ADC at Enable register */
  ret = I2C0_Transmit(TCS34727_ADDR, TCS34727_CMD|TCS34727_ENABLE_R_ADDR, TCS34727_ENABLE_PON |TCS34727_ENABLE_AEN);
  if(ret != 0)
    UART0_OutString("Error on Transmit\r\n");
  else
    UART0_OutString("TCS34727 RGBC On\r\n");

  //Integration Time Delay when Activating. Varies with Integration Time Choosen by User
  DELAY_1MS(3);

  UART0_OutString("TCS34727 Color Sensor Initialized\r\n");

}
```



```
/*  ---------------TCS34727_GET_RAW_CLEAR-------------
 *  Receive RAW clear data reading from the sensor
 *  Input: none
 *  Output: Returns 16-bit RAW clear data
 */
uint16_t TCS34727_GET_RAW_CLEAR(void){
  uint8_t CLEAR_LOW;
  uint8_t CLEAR_HIGH;
  uint16_t CLEAR_DATA;

  /* Use I2C to grab both HIGH and LOW data */
  CLEAR_LOW = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_CDATAL_R_ADDR);
  CLEAR_HIGH = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_CDATAH_R_ADDR);

  /* Concatanate into 16-bit value */
  CLEAR_DATA = (CLEAR_HIGH << 8) | CLEAR_LOW;

  //Integration Time Delay
  DELAY_1MS(3);

  return CLEAR_DATA;
}

/*  ---------------TCS34727_GET_RAW_RED---------------
 *  Receive RAW red data reading from the sensor
 *  Input: none
 *  Output: Returns 16-bit RAW red data
 */
uint16_t TCS34727_GET_RAW_RED(void){
```



```
  uint8_t RED_LOW;
  uint8_t RED_HIGH;
  uint16_t RED_DATA;

  /* Use I2C to grab both HIGH and LOW data */
  //CODE_FILL
  //uint8_t I2C0_Receive(uint8_t slave_addr, uint8_t slave_reg_addr)
  //#define TCS34727_RDATAL_R_ADDR          (CONSTANT_FILL)
  //#define TCS34727_RDATAH_R_ADDR          (CONSTANT_FILL)
  RED_LOW = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_RDATAL_R_ADDR);
  RED_HIGH = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_RDATAH_R_ADDR );


  /* Concatanate into 16-bit value */
  //CODE_FILL
  RED_DATA = (RED_HIGH << 8) | RED_LOW;

  //Integration Time Delay
  DELAY_1MS(3);

  return RED_DATA;
}

/*  ---------------TCS34727_GET_RAW_GREEN-------------
 *  Receive RAW green data reading from the sensor
 *  Input: none
 *  Output: Returns 16-bit RAW green data
 */
uint16_t TCS34727_GET_RAW_GREEN(void){
  uint8_t GREEN_LOW;
```



```
  uint8_t GREEN_HIGH;
  uint16_t GREEN_DATA;

  /* Use I2C to grab both HIGH and LOW data */
  //CODE_FILL
  //#define TCS34727_GDATAL_R_ADDR          (CONSTANT_FILL)
  //#define TCS34727_GDATAH_R_ADDR          (CONSTANT_FILL)
  GREEN_LOW = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_GDATAL_R_ADDR);
  GREEN_HIGH = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_GDATAH_R_ADDR);

  /* Concatanate into 16-bit value */
  //CODE_FILL
  GREEN_DATA = (GREEN_HIGH << 8) | GREEN_LOW;

  //Integration Time Delay
  DELAY_1MS(3);

  return GREEN_DATA;
}

/*  ---------------TCS34727_GET_RAW_BLUE-------------
 *  Receive RAW blue data reading from the sensor
 *  Input: none
 *  Output: Returns 16-bit RAW blue data
 */
uint16_t TCS34727_GET_RAW_BLUE(void){
  uint8_t BLUE_LOW;
  uint8_t BLUE_HIGH;
  uint16_t BLUE_DATA;
```

```
173   /* Use I2C to grab both HIGH and LOW data */
174   //CODE_FILL
175   //#define TCS34727_BDATAL_R_ADDR          (CONSTANT_FILL)
176   //#define TCS34727_BDATAH_R_ADDR
177   BLUE_LOW = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_BDATAL_R_ADDR);
178   BLUE_HIGH = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_BDATAH_R_ADDR);
179
180   /* Concatanate into 16-bit value*/
181   //CODE_FILL
182   BLUE_DATA = (BLUE_HIGH << 8) | BLUE_LOW;
183
184   //Integration Time Delay
185   DELAY_1MS(3);
186
187   return BLUE_DATA;
188 }
189
190 /*  ---------------TCS34727_GET_RGB-----------------
191  *  Normalize RAW data into RGB range (0-255)
192  *  Input: RGB Color Struct User Instance
193  *  Output: none
194  */
195 void TCS34727_GET_RGB(RGB_COLOR_HANDLE_t* RGB_COLOR_Instance){
196
197   /* Prevent Dividing by 0 by checking if the C_RAW value from struct is equal to 0 */
198   if(RGB_COLOR_Instance->C_RAW == 0){
199     RGB_COLOR_Instance->R = RGB_COLOR_Instance->G = RGB_COLOR_Instance->B = 0;
200     return;
201   }
202
```

```
203   /*
204   Divide all RGB value with their (RAW Value / Clear Raw Value) and multiple everything with 255.0
205   Store in RGB Color Instance Struct
206   */
207   float div_factor = 255.0 / (float)RGB_COLOR_Instance->C_RAW;
208
209   RGB_COLOR_Instance->R = ((float)RGB_COLOR_Instance->R_RAW * div_factor);
210   RGB_COLOR_Instance->G = ((float)RGB_COLOR_Instance->G_RAW * div_factor);
211   RGB_COLOR_Instance->B = ((float)RGB_COLOR_Instance->B_RAW * div_factor);
212
213 }
214
215 /*  ----------------Detect_Color--------------------
216  *  Detect which color is more prominant and returns that color
217  *  Input: RGB Color User Instance Struct
218  *  Output: COLOR_DETECTED enum value
219  */
220 COLOR_DETECTED Detect_Color(RGB_COLOR_HANDLE_t* RGB_COLOR_Instance){
221
222   /* Compare all values with eachother and return which color is prominent using enum type */
223   if (RGB_COLOR_Instance->R > RGB_COLOR_Instance->G && RGB_COLOR_Instance->R > RGB_COLOR_Instance->B) {
224       return RED_DETECT;
225   } else if (RGB_COLOR_Instance->G > RGB_COLOR_Instance->R && RGB_COLOR_Instance->G > RGB_COLOR_Instance->B) {
226       return GREEN_DETECT;
227   } else if (RGB_COLOR_Instance->B > RGB_COLOR_Instance->R && RGB_COLOR_Instance->B > RGB_COLOR_Instance->G) {
228       return BLUE_DETECT;
229   } else {
230       return NOTHING_DETECT;
231   }
232 }
```

ModuleTest.c

```
1  /*
2   * ModuleTest.c
3   *
4   *  Provides the testing functions all of individual peripheral testing
5   *  and full system testing
6   *
7   * Created on: September 3rd, 2023
8   *     Author: Jackie Huynh
9   *
10  */
11
12 #include "ModuleTest.h"
13 #include "TCS34727.h"
14 #include "MPU6050.h"
15 #include "UART0.h"
16 #include "Servo.h"
17 #include "LCD.h"
18 #include "I2C.h"
19 #include "util.h"
20 #include "ButtonLED.h"
21 #include "tm4c123gh6pm.h"
22 #include <stdio.h>
23 #include <string.h>
24 #include <stdint.h>
```

```
25
26  static char printBuf[100];
27  static char angleBuf[LCD_ROW_SIZE];
28  static char colorBuf[LCD_ROW_SIZE];
29  static char colorString[6];
30
31  char buffer[100];
32  int intValue = 10;
33  float floatValue = 3.14159;
34
35  /* RGB Color Struct Instance */
36  RGB_COLOR_HANDLE_t RGB_COLOR;
37
38  /* MPU6050 Struct Instance */
39  MPU6050_ACCEL_t Accel_Instance;
40  MPU6050_GYRO_t  Gyro_Instance;
41  MPU6050_ANGLE_t Angle_Instance;
42
43 static void Test_Delay(void){
44    LEDs=RED;
45    DELAY_1MS(500);
46    LEDs = DARK;
47    DELAY_1MS(500);
48  }
49
50 static void Test_UART(void){
51    /*CODE_FILL*/
52    // 1. Construct a string with letters, decimal numbers and floats using sprintf
53    // 2. Send the string to PC serial terminal for display
54    // 3. Delay for 1s using ms delay function
```

```
55    //UART0_OutString("Welcome to Bluetooth Controlled LED and DC Motor App");
56
57     /* Format and print various data */
58    sprintf(buffer, "Integer: %d, Float: %.2f", intValue, floatValue);
59    UART0_OutString(buffer);
60    UART0_OutCRLF();
61    DELAY_1MS(1000);
62
63  }
64
65 static void Test_I2C(void){
66    /*CODE_FILL*/
67    /* Check if RGB Color Sensor has been detected and display the ret value on PC serial terminal. */
68    uint8_t ret = I2C0_Receive(TCS34727_ADDR, TCS34727_CMD|TCS34727_ID_R_ADDR);
69      char buffer[20];  // Buffer to hold the formatted string
70   // Format the value of ret as a string
71      sprintf(buffer, "Received: %x\n", ret);
72
73      // Print the formatted string to the serial terminal
74      UART0_OutString(buffer);
75      UART0_OutCRLF();
76      DELAY_1MS(10);
77  }
78
79 static void Test_MPU6050(void){
80    /* Accelerometer and Gyroscope Data Instances */
81    MPU6050_ACCEL_t Accel_Instance;
82    MPU6050_GYRO_t Gyro_Instance;
83    MPU6050_ANGLE_t Angle_Instance;
84
```

```
85    /* Grab Accelerometer and Gyroscope Raw Data*/
86    MPU6050_Get_Accel(&Accel_Instance);
87    MPU6050_Get_Gyro(&Gyro_Instance);
88
89    /* Process Raw Accelerometer and Gyroscope Data */
90    MPU6050_Process_Accel(&Accel_Instance);
91    MPU6050_Process_Gyro(&Gyro_Instance);
92
93    /* Calculate Tilt Angle */
94    MPU6050_Get_Angle(&Accel_Instance, &Gyro_Instance, &Angle_Instance);
95
96    /* Format buffer to print data and angle */
97    char buffer[100];
98    sprintf(buffer, "Accel: X=%.2f Y=%.2f Z=%.2f, Gyro: X=%.2f Y=%.2f Z=%.2f, Angle: X=%.2f Y=%.2f Z=%.2f",
99                 Accel_Instance.Ax, Accel_Instance.Ay, Accel_Instance.Az,
100                Gyro_Instance.Gx, Gyro_Instance.Gy, Gyro_Instance.Gz,
101                Angle_Instance.ArX, Angle_Instance.ArY, Angle_Instance.ArZ);
102
103
104   /* Print to UART or any other output */
105   UART0_OutString(buffer);
106   UART0_OutCRLF();
107
108   DELAY_1MS(500);
109 }
110
111
112 static void Test_TCS34727(void){
113   /* Grab Raw Color Data From Sensor */
114   uint16_t B_Raw;
```

```
115   uint16_t G_Raw;
116   uint16_t R_Raw;
117   uint16_t C_Raw;
118
119   B_Raw = TCS34727_GET_RAW_BLUE();
120   G_Raw = TCS34727_GET_RAW_GREEN();
121   R_Raw = TCS34727_GET_RAW_RED();
122   C_Raw = TCS34727_GET_RAW_CLEAR();
123
124   /* Create RGB_COLOR_HANDLE_t instance */
125   RGB_COLOR_HANDLE_t RGB_COLOR_Instance;
126   RGB_COLOR_Instance.B_RAW = B_Raw;
127   RGB_COLOR_Instance.G_RAW = G_Raw;
128   RGB_COLOR_Instance.R_RAW = R_Raw;
129   RGB_COLOR_Instance.C_RAW = C_Raw;
130
131   /* Process Raw Color Data to RGB Value */
132   TCS34727_GET_RGB(&RGB_COLOR_Instance);
133
134
135   /* Change Onboard RGB LED Color to Detected Color */
136   switch(Detect_Color(&RGB_COLOR_Instance)){
137     case RED_DETECT:
138       LEDs = RED;
139       break;
140     case GREEN_DETECT:
141       LEDs = GREEN;
142       break;
143     case BLUE_DETECT:
144       LEDs = BLUE;
```

```
145       break;
146     case NOTHING_DETECT:
147       LEDs = DARK;
148       break;
149     }
150
151
152
153     /* Format String to Print */
154   char buffer[100]; // Assuming a sufficiently large buffer
155   float B_Float = (float)RGB_COLOR_Instance.B;
156   float G_Float = (float)RGB_COLOR_Instance.G;
157   float R_Float = (float)RGB_COLOR_Instance.R;
158
159   sprintf(buffer, "R: %.2f, G: %.2f, B: %.2f\n",
160           R_Float, G_Float, B_Float);
161   //sprintf(buffer, "R: %u, G: %u, B: %u\n",
162   //        RGB_COLOR_Instance.B_RAW,
163   //        RGB_COLOR_Instance.G_RAW,
164   //        RGB_COLOR_Instance.R_RAW);
165   //       // RGB_COLOR_Instance.C_RAW);
166
167
168   /* Print String to Terminal through USB */
169   UART0_OutString(buffer);
170   UART0_OutCRLF();
171   DELAY_1MS(1000);
172 }
173
174 static void Test_Servo(void){
```

```
175   /*
176    * In this test, follow the series of steps below (each step requires a 1s delay after)
177    * 1. Drive Servo to 0 degree
178    * 2. Drive Servo to -45 degree
179    * 3. Drive Servo to 0 degree
180    * 4. Drive Servo to 45 degree
181    * 5. Drive Servo to 0 degree
182    * 6. Drive Servo to -90 degree
183    * 7. Drive Servo to 0 degree
184    * 8. Drive Servo to 90 degree
185    */
186
187
188   Drive_Servo(0);       // Step 1
189   DELAY_1MS(1000); // Wait 1 second
190
191   Drive_Servo(-45);     // Step 2
192   DELAY_1MS(1000); // Wait 1 second
193
194   Drive_Servo(0);       // Step 3
195   DELAY_1MS(1000); // Wait 1 second
196
197   Drive_Servo(45);      // Step 4
198   DELAY_1MS(1000); // Wait 1 second
199
200   Drive_Servo(0);       // Step 5
201   DELAY_1MS(1000); // Wait 1 second
202
203   Drive_Servo(-90);     // Step 6
204   DELAY_1MS(1000); // Wait 1 second
```

```
205
206   Drive_Servo(0);       // Step 7
207   DELAY_1MS(1000); // Wait 1 second
208
209   Drive_Servo(90);      // Step 8
210   DELAY_1MS(1000);; // Wait 1 second
211 }
212
213 static void Test_LCD(void){
214
215   /* Print Name to LCD at Center Location */
216   /*CODE_FILL*/
217   LCD_Clear();  // Clear the LCD before printing anything new
218   DELAY_1MS(1000); // Short delay after clearing the LCD
219
220   // Centering "Ever" on a 16 character wide LCD
221   LCD_Set_Cursor(ROW1, 6);  // Set cursor to the first row, sixth column
222   LCD_Print_Str((uint8_t*)"Rod");
223   DELAY_1MS(1000);
224   LCD_Set_Cursor(ROW2, 5);
225
226   // Print "Ever" to the LCD
227   LCD_Print_Str((uint8_t*)"Agatep");
228   DELAY_1MS(1000); // Safety delay after writing to the LCD
229 }
230
231 static void Test_Full_System(void){
232   uint16_t B_Raw;
233   uint16_t G_Raw;
234   uint16_t R_Raw;
```

```
235   uint16_t C_Raw;
236
237   /* Accelerometer and Gyroscope Data Instances */
238   MPU6050_ACCEL_t Accel_Instance;
239   MPU6050_GYRO_t Gyro_Instance;
240   MPU6050_ANGLE_t Angle_Instance;
241
242   /* Grab Accelerometer and Gyroscope Raw Data*/
243   MPU6050_Get_Accel(&Accel_Instance);
244   MPU6050_Get_Gyro(&Gyro_Instance);
245
246   /* Process Raw Accelerometer and Gyroscope Data */
247   MPU6050_Process_Accel(&Accel_Instance);
248   MPU6050_Process_Gyro(&Gyro_Instance);
249
250   /* Calculate Tilt Angle */
251   MPU6050_Get_Angle(&Accel_Instance, &Gyro_Instance, &Angle_Instance);
252
253   /* Drive Servo Accordingly to Tilt Angle on X-Axis*/
254   Drive_Servo(Angle_Instance.ArX);
255
256   /* Format buffer to print data and angle */
257   char buffer2[100];
258   sprintf(buffer2, "Gyro: X=%.2f, Angle: X=%.2f",
259                Gyro_Instance.Gx,
260                Angle_Instance.ArX);
261
262
263   /* Print to UART or any other output */
264   UART0_OutString(buffer2);
```

```
265   UART0_OutCRLF();
266
267   /* Grab Raw Color Data From Sensor */
268
269   B_Raw = TCS34727_GET_RAW_BLUE();
270   G_Raw = TCS34727_GET_RAW_GREEN();
271   R_Raw = TCS34727_GET_RAW_RED();
272   C_Raw = TCS34727_GET_RAW_CLEAR();
273
274   /* Create RGB_COLOR_HANDLE_t instance */
275   RGB_COLOR_HANDLE_t RGB_COLOR_Instance;
276   RGB_COLOR_Instance.B_RAW = B_Raw;
277   RGB_COLOR_Instance.G_RAW = G_Raw;
278   RGB_COLOR_Instance.R_RAW = R_Raw;
279   RGB_COLOR_Instance.C_RAW = C_Raw;
280
281   /* Process Raw Color Data to RGB Value */
282   TCS34727_GET_RGB(&RGB_COLOR_Instance);
283
284   /* Change Onboard RGB LED Color to Detected Color */
285   switch(Detect_Color(&RGB_COLOR_Instance)){
286     case RED_DETECT:
287       LEDs = RED;
288       strcpy(colorString, "RED");
289       break;
290     case GREEN_DETECT:
291       LEDs = GREEN;
292       strcpy(colorString, "GREEN");
293       break;
294     case BLUE_DETECT:
```

```
      LEDs = BLUE;
      strcpy(colorString, "BLUE");
      break;
    case NOTHING_DETECT:
      LEDs = DARK;
      strcpy(colorString, "NA");
      break;
  }

  /* Format String to Print RGB value*/
char buffer[100]; // Assuming a sufficiently large buffer
float B_Float = (float)RGB_COLOR_Instance.B;
float G_Float = (float)RGB_COLOR_Instance.G;
float R_Float = (float)RGB_COLOR_Instance.R;

sprintf(buffer, "R: %.2f, G: %.2f, B: %.2f\n",
        R_Float, G_Float, B_Float);

  /* Print String to Terminal through USB */
  UART0_OutString(buffer);
  UART0_OutCRLF();
  //DELAY_1MS(1000);

  DELAY_1MS(200);

  /* Update LCD With Current Angle and Color Detected */

//  sprintf(angleBuf, "Angle:%0.2f\0", Angle_Instance.ArX);       //Format String to print angle to 2 Decimal Place
//  sprintf(colorBuf, "Color:%s\0", colorString);                 //Format String to print color detected
```

```
  sprintf(angleBuf, "Angle:%0.2f", Angle_Instance.ArX);       //Format String to print angle to 2 Decimal Place
  sprintf(colorBuf, "Color:%s", colorString);                 //Format String to print color detected

  /*CODE_FILL*/           //Clear LCD
  LCD_Clear();
  /*CODE_FILL*/           //Safety Delay of 2ms
  DELAY_1MS(2);
  /*CODE_FILL*/           //Set Cursor to Row 1 Column 0
    LCD_Set_Cursor(ROW1, 0);  // Set cursor to the first row, sixth column
  /*CODE_FILL*/           //Print angleBuf String on LCD
  LCD_Print_Str((uint8_t*)angleBuf);
  /*CODE_FILL*/           //Safety Delay of 2ms
  DELAY_1MS(2);
  /*CODE_FILL*/           //Set Cursor to Row 2 Column 1
  LCD_Set_Cursor(ROW2, 1);
  /*CODE_FILL*/           //Print colorBuf String on LCD
  LCD_Print_Str((uint8_t*)colorBuf);

  DELAY_1MS(20);
}
void Module_Test(MODULE_TEST_NAME test){

  switch(test){
    case DELAY_TEST:
      Test_Delay();
      break;

    case UART_TEST:
      Test_UART();
```

```
      break;

    case MPU6050_TEST:
      Test_MPU6050();
      break;

    case TCS34727_TEST:
      Test_TCS34727();
      break;

    case SERVO_TEST:
      Test_Servo();
      break;

    case LCD_TEST:
      Test_LCD();
      break;

    case FULL_SYSTEM_TEST:
      Test_Full_System();
      break;
    case I2C_TEST:
      Test_I2C();
      break;

    default:
      break;
  }

}
```

CODE EXPLANATION

The Test_Delay function briefly turns on an LED (presumably for visual feedback) using the LEDs variable and then waits for 500 milliseconds before turning off the LED. This delay of half a second is intended for testing purposes.

The Test_UART function demonstrates the use of UART (Universal Asynchronous Receiver-Transmitter) communication. It constructs a string containing an integer and a float using sprintf, sends the string over UART to a PC serial terminal for display, and includes a delay of 1 second.

The Test_I2C function demonstrates I2C (Inter-Integrated Circuit) communication. It checks if an RGB color sensor (likely the TCS34727) is detected using the I2C0_Receive function, formats the result into a string, and sends it over UART to a PC serial terminal for display, including a short delay of 10 milliseconds.

The Test_MPU6050 function tests an MPU6050 sensor, likely including an accelerometer and a gyroscope. It retrieves raw data from the accelerometer and gyroscope, processes the raw data into angles, and sends the calculated angles over UART for display, including a delay of 500 milliseconds.

The Test_TCS34727 function tests a TCS34727 color sensor. It retrieves raw color data from the sensor, processes it to obtain RGB values, controls an onboard RGB LED based on the detected

color, formats the RGB values into a string, and sends it over UART for display, including a delay of 1 second.

The Test_Servo function tests a servo motor. It drives the servo motor to various angles sequentially (0, -45, 0, 45, 0, -90, 0, 90) with a 1-second delay between each movement.

The Test_LCD function tests an LCD display. It prints a name ("Rod Agatep") at the center of the LCD display and includes a 1-second delay after clearing the display.

The Test_Full_System function tests multiple components of the system in sequence. It first tests the MPU6050 as in Test_MPU6050, then drives the servo motor based on the angle obtained from the MPU6050, tests the TCS34727 color sensor as in Test_TCS34727, and finally updates an LCD display with the angle and color detected.

Conclusion

We began this project by checking out the components: 16x2 LCD with I2C, the TCS, and MPU. We used the servo motor that was in our 211 kits. We used the starter code and completed DELAY and UART module. We then demoed I2C early and got extra credit. We then completed the module. We also demoed MPU and got extra credit. Finally, the servo and LCD modules were completed; then we completed the full system.

The servo portion was demoed early, and we got 1 extra credit point. We could not finish the LCD portion during class. However, we got the display to work after class ended so we recorded a video, submitted the code, and sent you an email about it. We then demoed it on the day the entire project was due. We then finished the entire system and demoed at the beginning of class.

The hardest part of this project was finding the hex values for I2C and coding the gyroscope calculations.

As you can see this was the most complex project we have done. The past 3 semesters of embedded systems has been a blast and very engaging experience. We will be using our TM4Cs in our Senior Project. We will be creating an automatic parking system!