<table>
<tr><td>Group Number</td><td colspan="2">42</td></tr>
<tr><td colspan="2">Student 1: Michelle Chuwindra ID #101721043</td><td rowspan="2">/10</td></tr>
<tr><td colspan="2">Student 2: Nicholas Mutton ID #101115853</td></tr>
<tr><td colspan="3">Lab Session: Monday 12:30PM - 2:30PM<br>Lab Supervisors: Dr. Siva Chandrasekaran<br><br>**(Note: Failure to fill this in correctly may result in lost marks)**</td></tr>
</table>

# EEE20003 – S2, 2019

## Experiment E.5 – LED Game

# PROGRAM LISTINGS

## MCPIO.h

```
/* mcpio.h
 *
 * MCP23008 I/O port expander header file
 */
#ifndef SOURCES_MCPIO_H_
#define SOURCES_MCPIO_H_

//register addresses from MCP23008 data sheet
enum RegisterAddress{
	IODIR	= 0x00,
	IPOL	= 0x01,
	GPINTEN		= 0x02,
	DEFVAL	= 0x03,
	INTCON	= 0x04,
	IOCON	= 0x05,
	GPPU	= 0x06,
	INTF	= 0x07,
	INTCAP		= 0x08,
	GPIO	= 0x09,
	OLAT	= 0x0A
};

/*
 * Set the direction of the individual bits of the port
 * input => direction = 1
 * output => direction = 0
 *
 * @param[in]: direction
 */
void SetDirection(unsigned direction);

/*
 * Set direction as input
 *
 * @param:-
 */
void SetInput();

/*
 * Set direction as output
 *
 * @param:-
 */
void SetOutput();

/*
 * Read pins that are set as inputs
 *
 * @param:-
 */
void ReadPin();

/*
 * Write pins that are set as outputs
 *
 * @param[in]: data
 */
void WritePin(unsigned data);

#endif /* SOURCES_MCPIO_H_ */
```

**MCPIO.cpp**

```cpp
/**
 ============================================================================
 * @file     mcpio.cpp
 * @brief    API for MCP23008 I/O port expander
 ============================================================================
 *      Author: USER
 */

#include "i2c.h"
#include "mcpio.h"

using namespace USBDM;

// Address (LSB = R/W bit)
static const unsigned I2C_ADDRESS = 0x20<<1;
static const unsigned I2C_SPEED   = 400*kHz;

// Declare I2C interface
I2c0 i2c{I2C_SPEED, I2cMode_Polled};


/*
 * Set the direction of the individual bits of the port
 * input => direction = 1
 * output => direction = 0
 *
 * @param: direction
 */
void SetDirection(uint8_t direction){
	static const uint8_t txData[] = {IODIR, direction};
	i2c.startTransaction();
	i2c.transmit(I2C_ADDRESS, sizeof(txData), txData);
	i2c.endTransaction();
}


/*
 * Set direction as input
 *
 * @param:-
 */
void SetInput(){
	static const uint8_t txData[] = {IODIR, 0xFF};
	i2c.startTransaction();
	i2c.transmit(I2C_ADDRESS, sizeof(txData), txData);
	i2c.endTransaction();
}


/*
 * Set direction as output
 *
 * @param:-
 */
void SetOutput(){
	static const uint8_t txData[] = {IODIR, 0x00};
	i2c.startTransaction();
	i2c.transmit(I2C_ADDRESS, sizeof(txData), txData);
	i2c.endTransaction();
}


/*
```

```
 * Read pins that are set as inputs
 *
 * @param:-
 */
void ReadPin(){
	static uint8_t data[] = {GPIO, };
	i2c.startTransaction();
	i2c.receive(I2C_ADDRESS, sizeof(data), data);
	i2c.endTransaction();
}


/*
 * Write pins that are set as outputs
 *
 * @param: data
 */
void WritePin(unsigned data){
	uint8_t txData[] = {GPIO, data};
	i2c.startTransaction();
	i2c.transmit(I2C_ADDRESS, sizeof(txData), txData);
	i2c.endTransaction();
}
```

**PITlab5.h**

```
/*
 * PITlab5.h
 *
 * PIT header file
 */
#ifndef SOURCES_PITLAB5_H_
#define SOURCES_PITLAB5_H_

/*
 * Initialize PIT configurations for PIT use
 *
 * @param:-
 */
void initializePIT();

/*
 * Adjust the duration of waits
 *
 * @param[in]: unsigned score
 */
void SpeedAdvance(unsigned score);

#endif /* SOURCES_PITLAB5_H_ */
```

## PITlab5.cpp

```
/**
 ==============================================================================
 * @file  PITlab5.cpp
 * @brief
 * Use PIT for delay
==============================================================================
 */
#include "hardware.h"
#include "pit.h"

using namespace USBDM;

// Connection mapping - change as required
// Led is assumed active-low
using Timer        = Pit;
using TimerChannel = Timer::Channel<0>;

/*
 * Initialize PIT configurations for PIT use
 *
 * @param:-
 */
void initializePIT() {
   // Enable PIT
   Timer::configure();

   // Check for errors so far
   checkError();
}

/*
 * Adjust the duration of waits
 *
 * @param[in]: unsigned score
 */
void SpeedAdvance(unsigned score)
{
       // Delay in ticks using channel 0
    // This is a busy-waiting loop!
       switch (score/50)
       {
       case 0:
       case 1: TimerChannel::delay(100*ms);
                    break;
       case 2: TimerChannel::delay(90*ms);
                    console.writeln("NOW LVL 2");
                    break;
       case 3: TimerChannel::delay(80*ms);
               console.writeln("NOW LVL 3");
                    break;
       case 4: TimerChannel::delay(70*ms);
               console.writeln("NOW LVL 4");
                    break;
       case 5: TimerChannel::delay(60*ms);
               console.writeln("MAXIMUM SPEED");
                    break;
       default: TimerChannel::delay(50*ms);
               console.writeln("MAXIMUM SPEED");
                    break;
       }
}
```

## Main.cpp

```
/**
 ============================================================================
 * @file     main.cpp
 * @brief    Demonstrates use of MMA845x Accelerometer over I2C
 *
 ============================================================================
 */
#include <math.h>
#include "system.h"
#include "derivative.h"
#include "hardware.h"
#include "i2c.h"
#include "mma845x.h"
#include "delay.h"
#include "mcpio.h"
#include "PITlab5.h"

// Allows access to USBDM library name-space
using namespace USBDM;

/*************************************************
 * Global objects representing hardware
 **************************************************/

// I2C interface
I2c0     i2c0;

// Accelerometer via I2C
MMA845x  accelerometer(i2c0, MMA845x::AccelerometerMode_2Gmode);

/**************************************************/

static constexpr int MAX_OFFSET = 2;
static constexpr int MIN_OFFSET = -2;

/**
 * Provides an offset value that may vary by up to +/- 1 on each call.
 *
 * @return offset value in range [MIN_OFFSET .. MAX_OFFSET]
 */
int randomWalk() {
   static int offset = 0;

   switch(rand()%2) {
      case 0: offset -= 1; break;
      case 1: offset += 1; break;
   }
   if (offset < MIN_OFFSET) {
      offset = MIN_OFFSET;
   }
   if (offset > MAX_OFFSET) {
      offset = MAX_OFFSET;
   }
   return offset;
}

/*
 *Calculates the player's score
 *
 *@param[in]: unsigned data
 *@return int score value
 */
int scores(unsigned data){
```

```
      static int functionscore = 0;

      if (data == 1 || data == 8)
      {
         console.writeln("You have lost the game.");
         console.write("You held on for ")
                .write(functionscore/10)
                .writeln(" seconds.");
         functionscore = 0;
      }
      else{
            functionscore++;
      }

      return functionscore;
}

/**
 * Report accelerometer values
 *
 * @param[in] accelerometer Accelerometer to use
 */
void report(MMA845x &accelerometer) {
   int accelStatus;
   int16_t accelX,accelY,accelZ;

   accelerometer.readAccelerometerXYZ(accelStatus, accelX, accelY, accelZ);
   console.setPadding(Padding_LeadingZeroes).setWidth(2).
         write("s=0x").write(accelStatus,Radix_16).
         setPadding(Padding_LeadingSpaces).setWidth(10).
         write(", aX=").write(accelX).
         write(", aY=").write(accelY);
}

int main() {
   console.writeln("Starting\n");
   console.write("Device ID = 0x").write(accelerometer.readID(), Radix_16).writeln("(should be
0x1A)");

   // Check if any USBDM error yet (constructors)
   checkError();
   report(accelerometer);
   console.write("Doing simple calibration\n"
         "Make sure the device is level!\n");
   waitMS(2000);

   if (accelerometer.calibrateAccelerometer() != E_NO_ERROR) {
      console.write("Calibration failed!\n");
      __asm__("bkpt");
   }

   // Make sure we have new values
   waitMS(100);
   console.write("After calibration\n");

   //Reset Output settings for GPIO
   SetOutput();
   WritePin(0x00);

   int ScoreValue = 0;
   int accelStatus;
   unsigned data = 0;
   int16_t accelX,accelY,accelZ;
   initializePIT();
```

```
    for(;;) {
            accelerometer.readAccelerometerXYZ(accelStatus, accelX, accelY, accelZ);
            //console.writeln(accelY);
            data = (accelY + 3400)/850;          // recalibrate manually
            data += randomWalk();
            //console.writeln(data);

            //turn on LEDs
            switch(data)
            {
            case 1: WritePin(1);
                                    break;
            case 2: WritePin(2);
                                    break;
            case 3: WritePin(4);
                                    break;
            case 4: WritePin(8);
                                    break;
            case 5: WritePin(16);
                                    break;
            case 6: WritePin(32);
                                    break;
            case 7: WritePin(64);
                                    break;
            case 8: WritePin(128);
                                    break;
            }

            //calculate score values
            ScoreValue = scores(data);

            //adjust duration of waits
            SpeedAdvance(ScoreValue);
    }
}
```

## BRIEF DESCRIPTION

The program operates as a game in which the player tries to balance the circuit to keep the edge LEDs from lighting up. It uses the accelerometer in the MK20 board to measure the difference in position of the circuit and a PIT module to adjust the delay between each loop. According to the tilt of the board on the Y axis, one of the bar-graph LEDs will light up. The program will then start counting how many seconds the player has managed to keep any of the LEDs at both ends of the LED bar-graph from lighting up.

An API controlling the MCP23008 I/O Expander (mcpio.h and mcpio.cpp) is developed to control the process of changing the direction of the individual bits of the port, reading bits from the pins, and writing bits onto pins. The mainline of the program is then developed to obtain the appropriate axis of the accelerometer to control the LED bar-graph. Functions defined in the API are used to write values to the LED bar-graph. Next, a provided function named randomWalk() is implemented to apply offsets onto the values written onto the LED bar-graph and a scoring function is developed to count how long the player has managed to not let the edge LEDs from illuminating.

A PIT module has been developed to adjust the delays between each cycle. As the player moves up from one level to another, the delays between cycles are decreased. This causes the game to be more vigilant in monitoring the changes in position detected by the accelerometer and implementing randomWalk(). The more often randomWalk() offset is added to the values detected by the accelerometer, the harder it will be to keep the edge LEDs from lighting up.

## MODULE DESCRIPTION

The program operates using the following features:

1. An API that controls the MCP23008 I/O Expander – mcpio.h and mcpio.cpp
2. The accelerator that measures the change in the Y axis
3. A PIT module that changes the duration of waits between cycles – PITlab5.h and PITlab5.cpp
4. The mainline that writes values onto the pins and counts the player scores (main.cpp)

### *MCP23008 I/O Expander API*

The API controls changing the direction of the individual bits on the port, reading bits from the pins, and writing bits onto the pins. This is done by using $I^2C$ protocols and reading from and writing onto the register addresses available on the port expander. Changing the direction of the bits on the port involves writing onto the IODIR register, and writing and reading from the LEDs require reading and writing from the GPIO register. The register addresses are specified in the header file (mcpio.h) and was derived from the provided data sheet for the port expander.

The mainline uses the functions defined in the API by sending the values of the data to be written onto the GPIO port. The API will then use I2C transmit function to send the data to the designated register – in this case, the GPIO register. Writing appropriate values onto the GPIO will in turn light up the appropriate bar-graph LEDs.

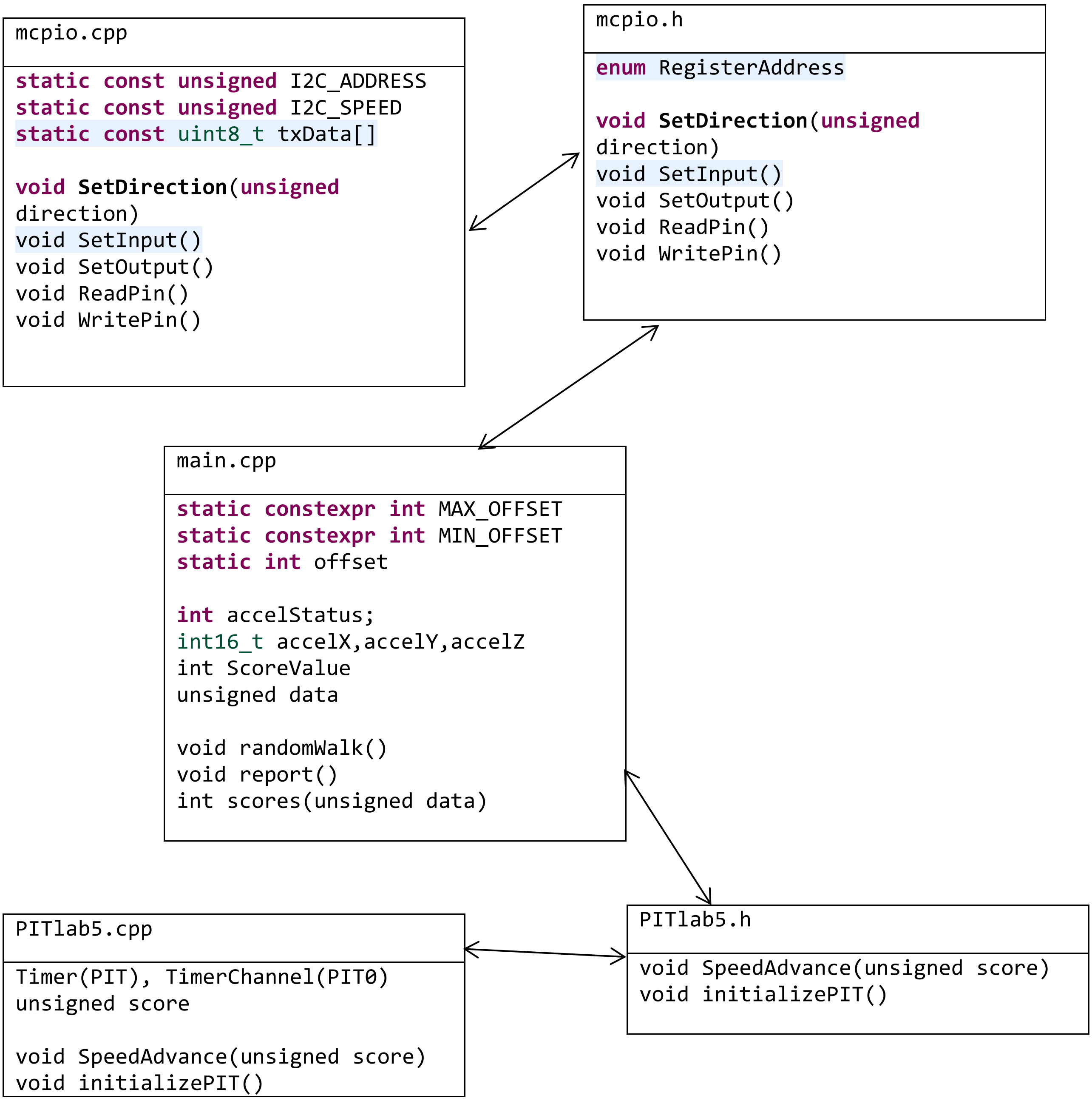
### *PIT Module*

The PIT module changes the delays between each cycle specified in the mainline. The module operates by defining the configuration of the PIT being used and using a switch case to decide which amount of delay is to be used in each cycle. The function takes in the number of cycles the player has managed to survive so far and when the number of cycles hits a certain threshold (in this case, every 50 cycles), the duration of waits between cycles is reduced. The PIT module defined a busy-wait timer, but in this context, it is enough to serve as a delay counter in the program.

### *Main Module*

The main module defines the use of the accelerometer and the I²C interface. It controls how the game operates by calling functions specified in both the I/O Expander API and the PIT module to write values onto the LED pins and define the duration of wait after each cycle. It first begins by checking the accelerometer and calibrating it with the function report(), then using the I/O Expander API to set the LED pins to outputs. This is followed by a switch case that determines which LED to turn on at various positions of the Y axis of the accelerometer. Manual re-calibration is used to divide the levels of tilt into 8 LED levels, and the I/O Expander API already contains the functions to be used to read and write onto pins with the I²C protocols. It then calculates the player's score with the scores() function that takes in the current tilt of the device from the accelerometer. Finally, the delay is adjusted to the score and difficulty of the game with the functions defined in the PIT module.

## DIAGRAM

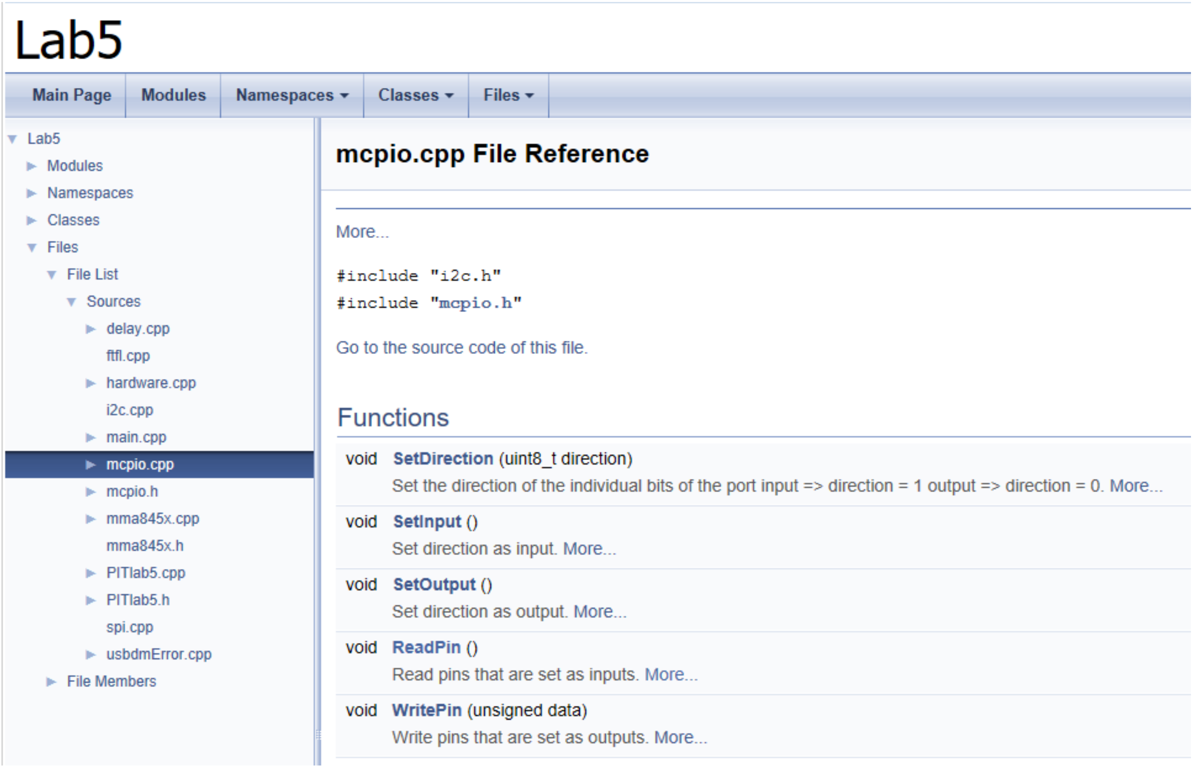


## VARIABLE TABLE

| Module | Variable Name | Type | Purpose |
|---|---|---|---|
| Main.cpp | MAX_OFFSET | static constexpr int | Maximum offset used in the RandomWalk() function to generate offsets for the LED game |
| | MIN_OFFSET | static constexpr int | Minimum offset used in the RandomWalk() function to generate offsets for the LED game |
| | offset | static int | Offset value returned by the randomWalk() function to modify the position of the LED in the game |
| | accelStatus | int | Parameter used in reading the values of the position changes of the device from the accelerometer |
| | accelX | int16_t | The value of position changes in the X axis of the device from the accelerometer |
| | accelY | int16_t | The value of position changes in the Y axis of the device from the accelerometer |
| | accelZ | int16_t | The value of position changes in the Z axis of the device from the accelerometer |
| | ScoreValue | int | The score of the player obtained from the scores() function |
| | data | unsigned | The value that controls which LED is to be lit |
| MCPIO.cpp | I2C_ADDRESS | static const unsigned | The address of the device being used |
| | I2C_SPEED | static const unsigned | The frequency of the device |
| | txData[] | static const uint8_t | The array of data to be written to or read from the pins |
| MCPIO.h | RegisterAddress | enum | Table of register addresses from the data sheet |

| | | | |
|---|---|---|---|
| PITlab5.cpp | score | unsigned | The player's current score used to determine the speed of the game / the delay between each cycles |

## DOXYGEN FOR GPIO API

# Lab5

Main Page | Modules | Namespaces | Classes | Files

- Lab5
  - Modules
  - Namespaces
  - Classes
  - Files
    - File List
      - Sources
        - delay.cpp
        - ftfl.cpp
        - hardware.cpp
        - i2c.cpp
        - main.cpp
        - mcpio.cpp
        - mcpio.h
        - mma845x.cpp
        - mma845x.h
        - PITlab5.cpp
        - PITlab5.h
        - spi.cpp
        - usbdmError.cpp
    - File Members

## mma845x.h File Reference

Interface for MMA845x accelerometer. More...

```
#include <stdint.h>
#include "i2c.h"
```

Go to the source code of this file.

### Classes

class **USBDM::MMA845x**
Class representing an interface for **MMA845x** 3-axis accelerometer over I2C. More...

### Namespaces

**USBDM**
Namespace enclosing **USBDM** classes.

### Detailed Description


# Lab5

Main Page | Modules | Namespaces | Classes | Files


- Lab5
  - Modules
  - Namespaces
  - Classes
  - Files
    - File List
      - Sources
        - delay.cpp
        - ftfl.cpp
        - hardware.cpp
        - i2c.cpp
        - main.cpp
        - mcpio.cpp
        - mcpio.h
        - mma845x.cpp
        - mma845x.h
        - PITlab5.cpp
        - PITlab5.h
        - spi.cpp
        - usbdmError.cpp
    - File Members

## mma845x.cpp File Reference

Created on: 22/11/2013 Author: podonoghue. More...

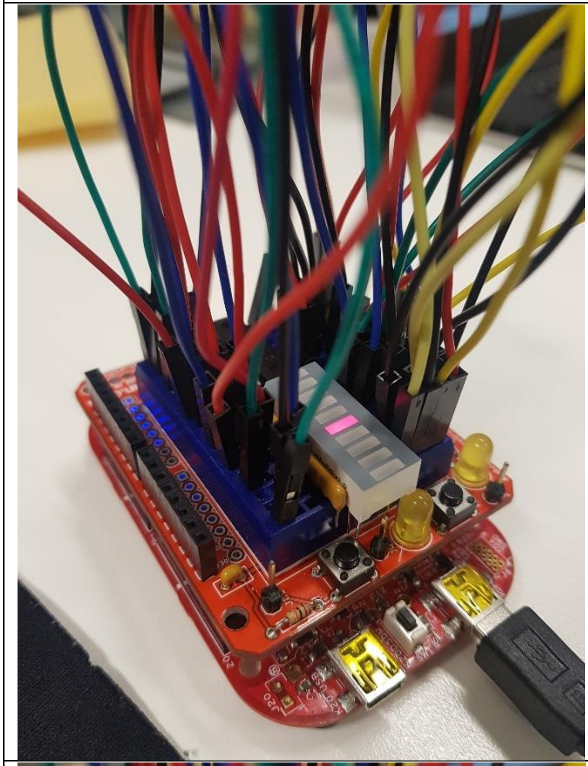
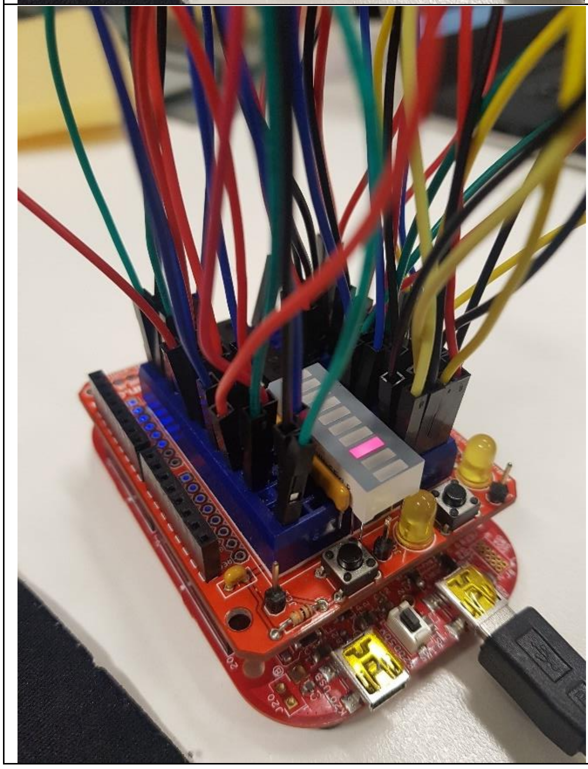
```
#include "mma845x.h"
#include "delay.h"
```

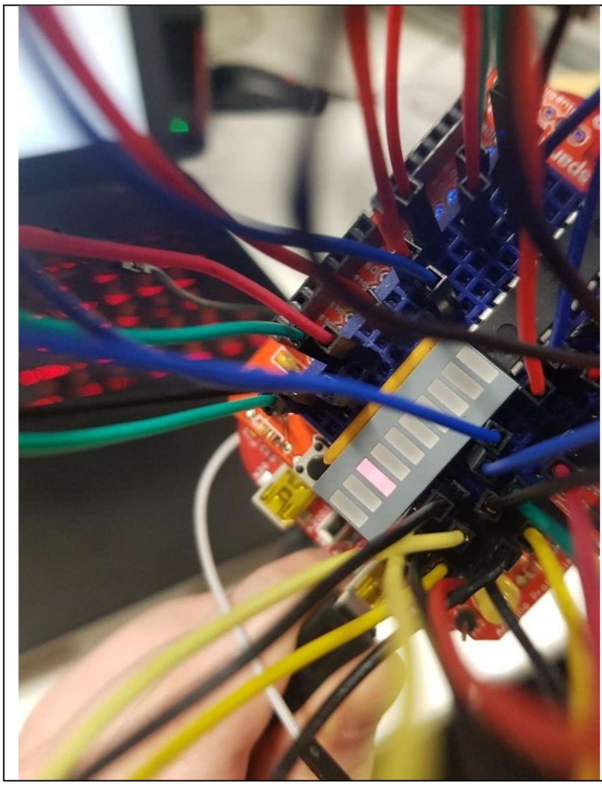
Go to the source code of this file.

### Macros

```
#define MMA845x_CTRL_REG1_ACTIVE_MASK (1<<0)
#define MMA845x_CTRL_REG1_F_READ_MASK (1<<1)
#define MMA845x_CTRL_REG1_LNOISE_MASK (1<<2)
#define MMA845x_CTRL_REG1_DR_OFF (3)
#define MMA845x_CTRL_REG1_DR_MASK (0x7<<MMA845x_CTRL_REG1_DR_OFF)
#define MMA845x_CTRL_REG1_DR(x) (((x)<<MMA845x_CTRL_REG1_DR_OFF)&MMA845x_CTRL_REG1_DR_MASK)
#define MMA845x_CTRL_REG1_ASLP_RATE_OFF (6)
#define MMA845x_CTRL_REG1_ASLP_RATE_MASK (0x3F<<MMA845x_CTRL_REG1_ASLP_RATE_OFF)
#define MMA845x_CTRL_REG1_ASLP_RATE(x) (((x)<<MMA845x_CTRL_REG1_ASLP_RATE_OFF)&MMA845x_CTRL_REG1_ASLP_RATE_MASK)
#define MMA845x_CTRL_REG2_MODS_OFF (0)
#define MMA845x_CTRL_REG2_MODS_MASK (0x3<<MMA845x_CTRL_REG1_DR_OFF)
#define MMA845x_CTRL_REG2_MODS(x) (((x)<<MMA845x_CTRL_REG1_DR_OFF)&MMA845x_CTRL_REG1_DR_MASK)
#define MMA845x_CTRL_REG2_SLPE_MASK (1<<2)
#define MMA845x_CTRL_REG2_SMODS_OFF (3)
#define MMA845x_CTRL_REG2_SMODS_MASK (0x3<<MMA845x_CTRL_REG1_DR_OFF)
#define MMA845x_CTRL_REG2_SMODS(x) (((x)<<MMA845x_CTRL_REG1_DR_OFF)&MMA845x_CTRL_REG1_DR_MASK)
#define MMA845x_CTRL_REG2_RST_MASK (1<<6)
#define MMA845x_CTRL_REG2_ST_MASK (1<<7)
```

**TESTING**

| RESPONSE | TEST CASES | EXPECTED RESULTS |
|---|---|---|
|  | Calibrating the accelerator<br><br>Device is placed on a flat surfaced and unmoved | The middle LED lights up and the console notifies that calibration is successful |
|  | Testing RandomWalk() function<br><br>Device is placed on a flat surface and left on its own for a few seconds | The position of the lit LED changes as randomWalk() offset changes the position of the lit LED every cycle |

| | | |
|---|---|---|
|  | Testing the accelerometer sensitivity<br><br>Tilting the device to the left/right | The LEDs on the left side light up when the device is tilted to the left, and vice versa |