Project 1 Report

CSE 398

Brandon Durrington
Ibrahima Diallo
Renjie Xu

Syracuse University School of Engineering and Computer Science

II. Table of Contents

*** Links in this document may not work on every platform. For easy navigation of the report visit: https://docs.google.com/document/d/1axGZ5tmygIqUHXNQoBMOcW4GbeBbRo99-4nVlMH3B_Y/edit?usp=sharing ***

III. Introduction

Project 1 is an introduction to embedded systems using Linux. We break down the implementation of the project into 7 seven steps (outlined in our Design and Implementation Plan section). We will be using a Raspberry Pi Zero W, a Liquid Crystal Display (LCD) Screen, a Color Sensor, and two push buttons.

The final state of the project will be a small embedded system with two modes. One mode that will read an object from a color sensor, determine its color ("red", "green", or "blue") and print that color onto an LCD display. If there is a red wire being shown to the sensor, then red will have the highest number value when we read the color data and "Red" will be displayed on the LCD screen.

The second mode will read an object from the sensor and print the "raw" data from the color neatly on the display. The raw data includes the red, green, blue, and clear values reported from the color sensor. If there is a red wire being shown then red will have the highest number value while the others are shown with lower values. This data will be shown in a neat format and the LCD output may look something like this:

```
"R: 224 G:126
 B:206   C:107"
```

IV. Design and Implementation Plan

We used Raspberry Pi Zero W and etched the operating system onto an SD card.. Then we were able to connect three USB's to it and from top to bottom it was implemented in this way: 1. USB that connects to USB Hub (provides power). 2. USB that connects to another USB Hub (which connects to keyboard and mouse). 3. HDMI cable that connects to the screen. All of our hardware interfacing was done through the Linux file handling system and we wrote C++ code on the Raspberry Pi's Geany.

Breakdown of Day-to-Day Signatures:

Day 1:

To show that we could get the Raspberry Pi OS working, the demonstration for Day 1 was to open the terminal in Raspberry pi desktop after the setup of Raspberry pi desktop. The point for Day1's work is 2 points.

Day 2:

The demonstration for Day 2 is to get direction and value at Pin4 by GPIO classes, change them by writing to the linux GPIO files, and get direction and value again on the terminal. This shows that we can write a program that writes to the Linux files properly in addition to the manual modification from the first signature. The point for Day2's work is 4 points.

Day 3:

To show that we can interface hardware and use it to change/read the values of GPIO pins, the demonstration for Day3 is to use a pushbutton to change the value of a GPIO input pin (between 0 and 1) and show that value on an LED. The point for Day3's work is 3 points.

Day 4:

To show that we can properly print to the LCD screen, the demonstration for Day4 is to print out "hello world" by the LCD screen. The point for Day4's work is 3 points.

Day 5:

The demonstration for Day5 is to show the color sensor working with output (print out RED, BLUE, GREEN) to the terminal based on the data that we read from the color sensor. The point for Day5's work is 3 points.

Day 6:

The demonstration for Day6 is to show the two push buttons switching between modes (one mode determining and printing the color read by the sensor, and the other for displaying the raw data). The point for Day6's work is 3 points.

Day 7:

The demonstration for Day7 is to show the whole project working that we use push buttons to switch modes and print out colors and the number of colors neatly on the LCD screen. The point for Day7's work is 2 points.

Summary of the contributions of the group members:

As a group, we set up the cables for the board and Raspberry pi zero W (connect to the screen) and debug NSIO programs together. Brandon sets up physical wires for led and pushbuttons, Ibrahima sets up physical wires for the LCD screen, and Renjie sets up physical wires for the color sensor. For the notebook, Brandon and Renjie maintain it. In the project, Brandon codes NSIO.h, and he also brainstorms the ASCII code for the LCD.cpp and how to make the color sensor print out the correct numbers. Ibrahima sets up the Raspberry pi desktop first, then he codes NSIO.cpp, GPIO.h, GPIO.cpp, main.cpp, LCD.h, LCD.cpp, Color.cpp and Color.h. Renjie writes the Lead Project Manager Assignment sheet and brainstorms the pushbutton part, the use of i2c tools and Adafruit color sensor board, and how to make the color sensor print out the correct numbers.

V. Description of Design Results

Here are the nine files we used when creating our project: main.cpp, NSIO.cpp, NSIO.h, GPIO.cpp, GPIO.h, LCD.cpp, LCD.h, Color.cpp, Color.h

main.cpp

The main.cpp is where we initialized all of our objects from the GPIO, NSIO, LCD and Color classes. This is where we ran most of our tests. Whenever we wanted to add a piece of hardware to the project in a certain step, we created a demonstration to verify that our code and hardware were properly implemented. The different demos are all in our main.cpp file, organized by block comments and labeled with single line comments.

We first tested that we could properly create a GPIO object, set its direction to out, set it's value to 1, and then set it's value to 0. With the code compiled successfully, we tested that the hardware worked by hooking up an led and 10kOhm resistor in a simple circuit to that specific GPIO port.

We tested that we could properly interface a pushbutton by creating a GPIO object and wiring a push button to it in a push button circuit like this. Our code ran a for loop that printed the value of the GPIO input pin every second for 10 seconds and we showed the value of the pin turning from a 0 to a 1 when the push button was pressed and back to 0 when the push button was no longer pressed.

We tested that we could properly interface and use an LCD object and its methods by creating an LCD object, running its .begin() method and .print("hello world").

Our color sensor demo created a Color (color sensor), called its .begin() method and ran a while loop. In our while loop we retrieved the values of four integers (red_data, blue_data, green_data, and clear_data) with our .readBytes() method and used them to determine what color was being shown to our sensor.

Our final code included an LCD object, a Color object, and two push buttons to determine the mode. If our mode was zero we ran the same loop from the color sensor demo and if the mode was 1, we called .printFour() to print out the raw data.

NSIO.cpp

NSIO.h

Header file for NSIO.cpp

GPIO.cpp

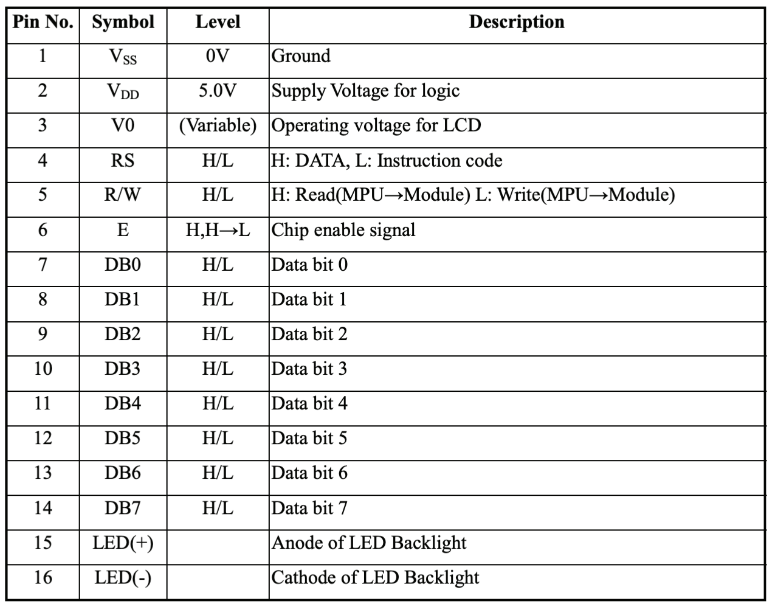
Our GPIO we were able to open up the "/sys/class/gpio/export" and add GPIO pins by changing the “in” and “out” of the pins and can also change the values of it then we close it in our functions that we made. Also used setters and getters to describe this implementation. Since when you are changing the values of the pins and the direction you are accessing /sys/class/gpio/gpio + pinNum and since we are only worried about the value and direction those are the only things we have to change. Like also when we are trying to add a pin and are trying to edit it we would just do "echo 4 >> export" if we are trying to edit the 4th pin number.

GPIO.h

Header file for GPIO.cpp.

LCD.cpp

Created the constructor with the eight Data Busses as well as the Register Selector(RS) and the Enable we all declared as objects of the GPIO file with the 0 being passed in as the value and it being for the out. Our methods we have in our cpp file includes print that takes in a string that will print out that string. By doing that we will have a method inside that will turn the address into ASCII code to print the data. Then for print we will loop through the letters of the ASCII and RS is 1 for making the letter and E is going high to low. For the commands method we put each Data Bus to take the command which is a 8 bit piece of data and set it with another bit to isolate that bit then we will shift it to make sure it is isolated.



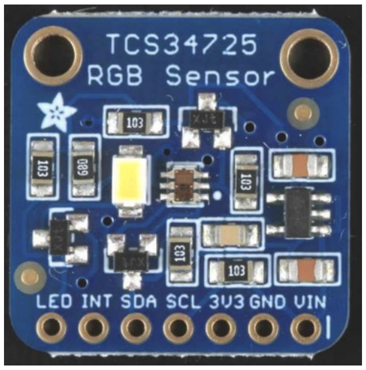
Pin No.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{DD}	5.0V	Supply Voltage for logic
3	V0	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA, L: Instruction code
5	R/W	H/L	H: Read(MPU→Module) L: Write(MPU→Module)
6	E	H,H→L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	LED(+)		Anode of LED Backlight
16	LED(-)		Cathode of LED Backlight

Pins 1, 3, 5, and 16 of our LCD were connected to Ground. Pins 2 and 15 were connected to a 3.3V power source. Every other pin was connected to a Raspberry Pi GPIO port.

LCD.h
Header file for LCD.cpp.

Color.cpp

In our Color.cpp we have methods to read and write data using I2C communication protocols. This part was tough so we made sure to heavily comment our code.
In our write method we simply follow the procedure to use the I2C reference document. In our readBytes() method we would write the address of the register that we wanted to read to the command register and then read the bytes that we are interested in. We set the buffer[0] equal to the command parameter which we needed to compute as a 16-bit data value so we did "return (int)((buffer[1] * 256) + buffer[0]);" to shift the bit of the "high" register 8 bits to the right. For each initialization we had to wait a certain amount of time for the hardware and as well setting up other implementations using "OurWrite(0b10000001, 0x00);" "OurWrite(0b10000000, 0b00000001);" "usleep(10000);" "OurWrite(0b10000000, 0b00000011);"



We only needed to wire four of the ports of our Adafruit color sensor. "VIN" was connected to a 3.3V power source. "GND" was connected to ground. The "SDA" and "SCL" pins for I2C communication were connected to pin 3 and pin 5 of the Raspberry Pi 40-pin header respectively. These pins are the I2C1 SDA and I2C1 SCL ports of the RPI Zero W.

Color.h
Header file for color.cpp.

VII. Assessment of Experimental Results

We used object oriented programming techniques to control our hardware in this project. The GPIO, LCD, and Color classes shown later in this document specify the implementation of object methods and use of variables for the GPIO pins, LCD screen, and color sensor. The two push buttons that we used in this project were handled with two GPIO input pins. This object oriented approach allowed us to break the project down into smaller manageable parts. For example, we did not need to consider all of the private pointers and variables of the LCD class once implemented, we could treat it as a black box and once we knew that everything was working and use the public methods that we created to get the results that we want. Programming the project like this also made the implementation very easy to understand and I would recommend and object oriented approach for anyone looking to replicate this project or create something similar

For the GPIO pins we were able to use Linux to see the GPIO pins and make changes when necessary. We implemented that in our code and the GPIO class ended up being very useful to us when programming the other pieces of hardware.

When adding the LCD screen to the project, we were able to use our GPIO class to create a GPIO object for each of the lines of our LCD (register select, enable, etc.). This was useful as we already had constructors, getters, and setters for each line and were able to control every bit that was being sent to the LCD screen.

The Color sensor was a little difficult at times. It was not working because of how we passed our address using ioctl. We did not use the right slave address but after we got our slave address it was pretty simple. We were able to compare the values that we read from the Adafruit Color sensor and print out the largest color value. Using this implementation we put the two push buttons to switch modes where one mode prints one color that is being sensed the most while the other skips to the next line and adds more colors with the raw data of each color so all colors are printed at the exact same time. Eventually with the Color sensor implemented we were able to communicate with the LCD by wiring them together and using the addresses of the Color sensor and pass it through to the LCD.

We were able to finish our lab as we introduced it but there were hiccups that we were able to learn from. Hiccups like understanding how the hardware communicates using the I2C protocol and the GPIO layout wiring to specific areas.

VIII. Appendix

main.cpp:

```
// main.cpp

#include "NSIO.h"
#include "GPIO.h"
#include "LCD.h"
#include "Color.h"
#include <iostream>
#include <unistd.h>

int main () {

        //NSIO class demo
        /*
        NSIO pin1;
        //NSIO pin2(1, "out");

        pin1.setString("out");

        cout << "Watch" << endl;
        cout << pin1.getString() << endl;
        */

        //GPIO class demo
        /*
        GPIO pin4(4);
        pin4.setDirection("out");

        cout << "Direction: " << pin4.getDirection() << endl;
        cout << "Value: " << pin4.getVal() << endl;

        pin4.setVal(1);

        cout << "Direction: " << pin4.getDirection() << endl;
        cout << "Value: " << pin4.getVal() << endl;

        usleep(1000000); //1 sec usleep is in microseconds
        pin4.setVal(0);

        cout << "Direction: " << pin4.getDirection() << endl;
        cout << "Value: " << pin4.getVal() << endl;

        */

        //Pushbutton demo
        // /*
        //GPIO pin4(4, 1, "in");
        GPIO pin4(4);

        //Print value every second for 10 seconds
        for(int i=0; i < 10; i++) {
                cout << "Value: " << pin4.getVal() << endl;
                usleep(1000000);
        }
        // */

        //Print "hello world" on LCD demo
        /*
        LCD ourLCD(13, 26, 10, 12, 6, 5, 25, 24, 23, 22, 27);

        ourLCD.begin();
        ourLCD.print("hello world");
        */

        //Previous Color Sensor Demo
        //Now Final Project Code
        /*
        Color ourSensor(1);
        ourSensor.start(0x29);
        LCD ourLCD(13, 26, 10, 12, 6, 5, 25, 24, 23, 22, 27);
        ourLCD.begin();

        int red_data, green_data, blue_data, clear_data;
        int pMode = 0; //0: Display Color  1: Raw Data
        GPIO pin4(4, 0, "in"); //Pushbutton 1
        GPIO pin14(14, 0, "in"); //Pushbutton 2


        //Get and print color every second for 5 seconds
        //for(int i=0; i < 5; i++) {
        while(1) {

                //Check for mode
                    if(pin4.getVal() == 1) { pMode = 0; }
                else if(pin14.getVal() == 1) { pMode = 1; }

                //Get values
                red_data = ourSensor.readBytes(0x96);
                green_data = ourSensor.readBytes(0x98);
                blue_data = ourSensor.readBytes(0x9A);
                clear_data = ourSensor.readBytes(0x94);

                //Behavior based on mode
                if(pMode == 0) {
                        //Determine which is largest
                        if((red_data > green_data) && (red_data > blue_data)) {
                                cout << "RED" << endl;
                                ourLCD.clearDisp();
                                ourLCD.print("RED");

                        }
                        else if((green_data > red_data) && (green_data > blue_data)) {
                                cout << "GREEN" << endl;
                                ourLCD.clearDisp();
                                ourLCD.print("GREEN");
                        }
                        else if((blue_data > red_data) && (blue_data > green_data)) {
                                cout << "BLUE" << endl;
```

```
					ourLCD.clearDisp();
					ourLCD.print("BLUE");
				}
			}
			else {
				//Print raw data
				ourLCD.printFour(red_data, green_data, blue_data, clear_data);
				//cout << "RED: " << red_data << endl;
				//cout << "GREEN: " << green_data << endl;

			}

			usleep(1000000); //1 second
		}
		*/

		return 0;
}
```

NSIO.cpp:

```
//NSIO.cpp

#include "NSIO.h"

NSIO::NSIO() { num = 0; strng = "in"; }

NSIO::NSIO(int n, string s) {
	num = n;
	strng = s;
};

NSIO::~NSIO() {}

int NSIO::getNum() { return num; }
string NSIO::getString() { return strng; }
void NSIO::setNum(int n) { num = n; }
void NSIO::setString(string s) {strng = s; }
```

NSIO.h:

```
//NSIO.h

#ifndef _NSIO_H_
#define _NSIO_H_
#include <iostream>

using namespace std;

class NSIO {
	private:
		int num;
		string strng;
	public:

		NSIO();
		NSIO(int n, string s);
		~NSIO();
		int getNum();
		string getString();
		void setNum(int n);
		void setString(string s);
};

#endif
```

GPIO.cpp:

```
//GPIO.cpp

#include "GPIO.h"
#include <fstream>

// Our "default" constructor
GPIO::GPIO(int pNum) {
	pinNum = pNum;

	//Make the gpio_ file
	ofstream gpio;
	gpio.open("/sys/class/gpio/export");
	gpio << pinNum;
	gpio.close();

	setVal(0);
	setDirection("in");
}

//Parameterized constructor
GPIO::GPIO(int pNum, int v, string dir) {
	pinNum = pNum;

	//Make the gpio_ file
	ofstream gpio;
	gpio.open("/sys/class/gpio/export");
	gpio << pinNum;
	gpio.close();

	setVal(v);
	setDirection(dir);
};

//This destructor should set the pins to how they were found
GPIO::~GPIO() {
	/*
	 * Set pin val to 0
	 * Direction to "in"
	 * and "unexport" the allocated GPIO file folders
	*/
```

```
		setVal(0);
		setDirection("in");
		ofstream gpio;
		gpio.open("sys/class/gpio/unexport");
		gpio << pinNum;
		gpio.close();
}

/**************** GETTERS ****************/

int GPIO::getPinNum() { return pinNum; }

int GPIO::getVal() {
		int val;
		ifstream value;
		value.open("/sys/class/gpio/gpio" + to_string(pinNum) + "/value");
		value >> val;
		value.close();
		return val;
}

string GPIO::getDirection() {
		string dir;
		ifstream direction;
		direction.open("/sys/class/gpio/gpio" + to_string(pinNum) + "/direction");
		direction >> dir;
		direction.close();
		return dir;
}


/**************** SETTERS ****************/

void GPIO::setVal(int v) {
		ofstream value;
		value.open("/sys/class/gpio/gpio" + to_string(pinNum) + "/value");
		value << v;
		value.close();
}


void GPIO::setDirection(string dir) {
		ofstream direction;
		direction.open("/sys/class/gpio/gpio" + to_string(pinNum) + "/direction");
		direction << dir;
		direction.close();
}
```

GPIO.h:

```
//GPIO.h

#ifndef _GPIO_H_
#define _GPIO_H_
#include <iostream>
```

```
using namespace std;

class GPIO {
	private:
		int pinNum;

	public:

		GPIO(int pNum);
		GPIO(int pNum, int v, string dir);
		~GPIO();
		int getPinNum();
		int getVal();
		string getDirection();
		void setVal(int v);
		void setDirection(string dir);

};

#endif
```

LCD.cpp:

```
// LCD.cpp

#include "LCD.h"
#include "GPIO.h"
#include <unistd.h>

using namespace std;

LCD::LCD(int Eparam, int RSparam, int RWparam, int D0param, int D1param, int D2param,
			int D3param, int D4param, int D5param, int D6param, int D7param) {
	//Create GPIO pins
	//Set directions to "out"
	//Set values to 0
	E = new GPIO(Eparam);
	RS = new GPIO(RSparam);
	RW = new GPIO(RWparam);
	D0 = new GPIO(D0param);
	D1 = new GPIO(D1param);
	D2 = new GPIO(D2param);
	D3 = new GPIO(D3param);
	D4 = new GPIO(D4param);
	D5 = new GPIO(D5param);
	D6 = new GPIO(D6param);
	D7 = new GPIO(D7param);
	E->setDirection("out");
	RS->setDirection("out");
	RW->setDirection("out");
	D0->setDirection("out");
	D1->setDirection("out");
	D2->setDirection("out");
```

```
        D3->setDirection("out");
        D4->setDirection("out");
        D5->setDirection("out");
        D6->setDirection("out");
        D7->setDirection("out");
        E->setVal(0);
        RS->setVal(0);
        RW->setVal(0);
        D0->setVal(0);
        D1->setVal(0);
        D2->setVal(0);
        D3->setVal(0);
        D4->setVal(0);
        D5->setVal(0);
        D6->setVal(0);
        D7->setVal(0);


}

LCD::~LCD() {
        delete E;//Enable
        delete RS;        //Register Select
        delete RW;
        delete D0;        //Data Bus 0
        delete D1;        //Data Bus 1
        delete D2;        //Data Bus 2
        delete D3;        //Data Bus 3
        delete D4;        //Data Bus 4
        delete D5;        //Data Bus 5
        delete D6;        //Data Bus 6
        delete D7;        //Data Bus 7

}

//Set register select, enable, and Data bus 0-7 values with an integer
//Changes DDRAM adress to move cursour
void LCD::command(int instruct) {
        //RS will be 0 for moving the cursor
        RS-> setVal(0);
        RW-> setVal(0);

        //and bits to get each DB value
        D0-> setVal((instruct & 1));
        D1-> setVal((instruct & 2) >> 1);
        D2-> setVal((instruct & 4) >> 2);
        D3-> setVal((instruct & 8) >> 3);
        D4-> setVal((instruct & 16) >> 4);
        D5-> setVal((instruct & 32) >> 5);
        D6-> setVal((instruct & 64) >> 6);
        D7-> setVal((instruct & 128) >> 7);

        //Enable high to low
        usleep(50);
        E-> setVal(1);
        usleep(50);
        E-> setVal(0);
        usleep(1600);

}

void LCD::clearDisp() {
        //Clear display
        command(0x01);

        //Move cursor to start
        command(0x02);
}

void LCD::print(string letters) {

        //RS is 1 for making letter
        RS-> setVal(1);
        RW-> setVal(0);

        //Put ascii code on the data bus gpio objects
        //Loop through letters
        //Get ASCII from each letter for D0-7
        for(int i=0; i<letters.length(); i++) {
                        char lChar = letters.at(i);
                        int asciiVal = (int)lChar;
                        D0-> setVal((asciiVal & 1));
                        D1-> setVal((asciiVal & 2) >> 1);
                        D2-> setVal((asciiVal & 4) >> 2);
                        D3-> setVal((asciiVal & 8) >> 3);
                        D4-> setVal((asciiVal & 16) >> 4);
                        D5-> setVal((asciiVal & 32) >> 5);
                        D6-> setVal((asciiVal & 64) >> 6);
                        D7-> setVal((asciiVal & 128) >> 7);
                        //E goes high low
                        usleep(50);
                        E-> setVal(1);
                        usleep(50);
                        E-> setVal(0);
        }
}

void LCD::printFour(int red_data, int green_data, int blue_data, int clear_data) {
        //Clear display
        command(0x01);

        //RS is 1 for making letter
        RS-> setVal(1);
        RW-> setVal(0);

        //Print red @ DDRAM 00
        command(0x00);
        print("R:" + to_string(red_data));

        //Print green @ DDRAM 08
        command(0x06);
        print("G:" + to_string(green_data));
        //Print blue @ DDRAM 40
        command(0xC0);
        print("B:" + to_string(blue_data));

        //Print clear @ DDRAM 48
        command(0xC8);
        print("C:" + to_string(clear_data));


        //E goes high low
        usleep(50);
        E-> setVal(1);
        usleep(50);
        E-> setVal(0);

}

//Give time for the LCM to initialize
void LCD::begin() {

        //Wait 50ms; should be enough time; check data sheet seciton 14
        usleep(5000);

        command(0b00111000);
        command(0x38);
        command(0x0F);
        command(0x01);
        command(0x06);

}
```

LCD.h:

```
//LCD.h

#ifndef LCD_H
#define LCD_H

#include "GPIO.h"
#include <string>

using namespace std;

class LCD {
        public:
                LCD(int E, int RS, int RW, int D0, int D1, int D2,
                        int D3, int D4, int D5, int D6, int D7);
                ~LCD();

                void clearDisp();
                void print(string letters);
                void printFour(int red_data, int green_data, int blue_data, int clear_data);
                void command(int command);
```

```
		void begin();

	private:
		GPIO* E;		//Enable
		GPIO* RS;		//Register Select
		GPIO* RW;	//Register
		GPIO* D0;		//Data Bus 0
		GPIO* D1;		//Data Bus 1
		GPIO* D2;		//Data Bus 2
		GPIO* D3;		//Data Bus 3
		GPIO* D4;		//Data Bus 4
		GPIO* D5;		//Data Bus 5
		GPIO* D6;		//Data Bus 6
		GPIO* D7;		//Data Bus 7


};
```

Color.cpp:

```
//Color.cpp

#include "Color.h"
#include "GPIO.h"
#include <unistd.h> //for usleep()
#include <linux/i2c-dev.h>

#include <sys/ioctl.h> //for i2ctools
#include <fcntl.h>
#include <errno.h> //for error codes

#include <stdio.h>

using namespace std;

int fd;

Color::Color(int bus) {
	fd = open("/dev/i2c-1", O_RDWR); //Get the file descriptor //Bus hardcoded to 1
}

Color::~Color() {
	//Worksheet 6 Question 9
}

void Color::OurWrite(int command, int val) {
	char buffer[2];
	buffer[0] = command;
	buffer[1] = val;
	int something = write(fd, buffer, 2);
}

int Color::readBytes(int command) {
	char buffer[2];
```

```
        buffer[0] = command;

        //Write the address to the command register
        write(fd, buffer, 1);

        //Read the two bytes
        read(fd, buffer, 2);

        //int color_data = (int)((buffer[1] * 256) + buffer[0]);

        return (int)((buffer[1] * 256) + buffer[0]);
        //return buffer[2];

}

//Always have this in main when using color sensor
void Color::start(int address) {

        /* Error checking
        //Checking if file opened correctly
        if(fd < 0) {
                printf("Error opening file: %s\n", strerror(errno)); //Print error #
                return;
        }

        //Checking for iotcl error
        if(ioctl(fd, 0x29,address) < 0) {
                printf("ioctl: %s\n", strerror(errno)); //Print error #
                return;
        }
        */

        //Start the i2c bus with a specific secondary device address
          ioctl(fd,I2C_SLAVE, 0x29);
        //ioctl(fd, i2c_slave, i2c_addr)

        //Set the integration time
        OurWrite(0b10000001, 0x00);

        //PON bit to power on color sensor
        OurWrite(0b10000000, 0b00000001);
        //Need to wait 2.4ms, we do 10ms here JIC.
        usleep(10000);
        //AEN bit
        OurWrite(0b10000000, 0b00000011);

        return;
}
```

Color.h:

```
//Color.h
```

```
#ifndef Color_H
#define Color_H

#include "GPIO.h"
#include <string>

using namespace std;

class Color {
	public:
		Color(int bus);
		~Color();

		void start(int address);
		void OurWrite(int command, int val);
		int readBytes(int command);

	private:

		//File Descriptor
		int fd;

};

#endif
```