SENIOR DESIGN PROJECT REPORT

ECET 49100 – Phase II

PC Systems II Lab(s) Android and .Netduino Communication via Bluetooth 2.0

Rachael Libby

Department of Electrical and Computer Engineering Technology
Purdue School of Engineering and Technology
Indiana University – Purdue University Indianapolis
April 04, 2017
ABSTRACT

This report will discuss the senior design project phase II. The report includes the concept, design, and testing of an Android application that sends a command via Bluetooth to a .Netduino board, which reads a temperature, pressure, and altitude chip and then displays those values on the Android phone. The Android application also controls LEDs on the .Netduino board. Upon completion of the senior design project, the student will have deliverables that consist of source code, lecture material, and lab procedures, which will be used for ECET 434 – PC Systems II.
PREFACE

This report is written in partial fulfillment of the requirements of ECET 491 Senior Project Phase II. This project was chosen by a recommendation of two professors in the Engineering Technology Department. I was interested in this project because I get to further my knowledge in both Android and the .NET environment, along with interest in how hardware and software communicate. By providing the lab write up and recommended lecture material, I feel confident in my comprehension of how both sections work which in turn credits my competence to convey directions to future lab students. Professor Phil Pash has been mentoring as well as helping me expand my abilities in the .NET environment. Professor Elizabeth Freije has also aided in this project.
TABLE OF CONTENTS
I. INTRODUCTION ................................................................................................................... 5
II. HARDWARE DESIGN ........................................................................................................... 6
   A. Major Components ........................................................................................................ 6
      1. Mainboard ................................................................................................................... 7
      2. Pressure Sensor ......................................................................................................... 7
      3. Bluetooth Shield ....................................................................................................... 9
III. SOFTWARE DESIGN ............................................................................................................ 9
   A. Netduino Software ........................................................................................................ 9
      1. Complier and Environment ..................................................................................... 9
      2. Code ........................................................................................................................... 10
   B. Android Software ..................................................................................................... Error! Bookmark not defined.
IV. TESTING ............................................................................................................................... 30
III. CONCLUSIONS AND RECOMMENDATIONS .................................................................... 31
LIST OF REFERENCES ................................................................................................................. 32
APPENDIX A. ITEAD BLUETOOTH DATASHEET ......Error! Bookmark not defined.
I. INTRODUCTION

The purpose of this report is to document the process, beginning to end, from concept through testing, of creating one or more labs for the PC Systems II course taught at IUPUI in the department of Engineering Technology. The purpose of the project serves two purposes, to demonstrate skills learned through the four years spent taking classes in the Engineering Technology department, the second being to create new lab assignment(s) for instructors to implement into the coursework agenda for upcoming semesters.

The idea was originally proposed to the student by two professors. One of whom often teaches the class and the other who coordinates the class. This led to well-defined expectations as well as assistance when the student had any questions pertaining to the design and implementation of the lab.

The goal of this project is to create a lab or set of labs that creates an Android application, which receives data from a .Netduino application and then sends the temperature, pressure, and altitude from the BMP180 sensor. The .Netduino app will send via Bluetooth the information when the Android user selects the “Get Temperature” button. The Android app will display, in a window, the temperature, pressure, and altitude. In addition, The Android app will consist of a few buttons, which via Bluetooth
will control red, yellow, and green LEDs on the .Netduino. Each color has a single button to turn on the corresponding LED as well as a “Stop Light” button which lights the LEDs in a stop light color pattern (red, green, yellow) while implementing a timer to control when the LEDs change color.

Upon completion of the senior project, the student must design, implement, and test the lab. The student may also make recommendations for modifications, which will seek approval from the instructor. In addition, the senior student must submit a final project that consists of a block diagram of the applications on both Android and .Netduino, source code of both applications, recommended modifications of the Bluetooth shield, communication protocol between the two devices, procedure for the lab, and recommended lecture material for preparation of the lab to the lab student.

II. HARDWARE DESIGN

A. Major Components

This section describes each component's function in the project.
1. **Mainboard**

The base of this project uses a .Netduino 3 Wi-Fi board whose microcontroller is a STMicro STM32F4. It has speeds of up to 168 MHz with code storage of up to 1408 KB. It uses the .NET Micro Framework 4.3 and contains 22 general-purpose input/output (GPIO) ports, 6 pulse width modulators (PWM), an I2C bus as well as SPI bus. This .Netduino board allows shields to be mounted on top of the .Netduino board in order to be more functional and versatile.

2. **Pressure Sensor**
The pressure sensor is an Adafruit BMP 180 Barometric Pressure / Temperature / Altitude sensor. This precision sensor from Bosch is the best low-cost sensing solution for measuring barometric pressure and temperature. Because pressure changes with altitude, it can also be used as an altimeter. The sensor is soldered onto a PCB with a 3.3V regulator, I2C level shifter and pull-up resistors on the I2C pins. The XCLR pin is not physically present on the BMP180 so if you need to know that data is ready you will need to query the I2C bus. This board is 5V compliant - a 3.3V regulator and a I2C level shifter circuit is included so the sensor can be safely used with 5V logic and power. To use with the microcontroller, simply connect the VIN pin to the 5V voltage pin, GND to ground, SCL to I2C Clock (Analog 5) and SDA to I2C Data (Analog 4). Then download the BMP085/BMP180 library.
3. Bluetooth Shield

Bluetooth Shield V2.2 is a serial port Bluetooth module (with master and slave mode) breakout board. It can directly plug on Arduino/IFlat-32 board and use UART ports for communication with Arduino/IFlat-32 or PC. It uses microprocessor CSR BC417 and follows UART/Bluetooth 2.0 communication protocol. When power on and disconnect the port, states LED blinks 1time/2s; when the module connect and open the serial port, states LED blinks 2times/s.

III. SOFTWARE DESIGN

A. Netduino Software

1. Complier and Environment

   Software for the .Netduino application was written entirely in C# language. The complier used was made by Microsoft. More information can be found on
their website at: www.visualstudio.com/vs. Visual Studio 2015 (VS15) enables one to write code accurately and efficiently without losing the current file context. One can easily zoom into details such as call structure, related functions, check-ins, and test status. Once can also leverage our functionality to refactor, identify, and fix code issues.

2. **Code**

Students must first begin by initializing the I2C pin. This consists of setting the LEDs as outputs and assigning each LED to the correct pin. Next, the initializing of the BMP180 sensor should occur. Once this has happened, the Bluetooth serial port should open and send/receive data. Inside of the serial port, the sensor reads pressure, temperature, and altitude and converts to standard units. Once the temperature readings have been taken and assigned to a variable, students will encode the string and data that will be sent out to the Android phone. In the Bluetooth event handler, the .Netduino receives commands from the Android app and uses a switch case to change the LED colors. This will occur, for example, when the user presses the red button on the Android phone, the letter ‘R’ is sent via Bluetooth to the .Netduino application. The .Netduino
application receives this letter and encodes it into a character and then uses that character to turn on the red LED.
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.Netduino;
using System.IO.Ports;

namespace BlueTooth_HC_05
{
    public class Program
    {
        static readonly double FEET_PER_METER = 3.28084;
        static bool get_Weather = false;
        private static OutputPort red, yellow, green; // create private static for the red led output
        private static SerialPort rs232_Port; // create private static for my RS Port
        static byte[] outBuffer = new byte[30];

        public static void Main()
        {
            // write your code here
            red = new OutputPort(Pins.GPIO_PIN_D7, false); // here i am assigning variable with there proper Pin out numbers and setting the output as off
            yellow = new OutputPort(Pins.GPIO_PIN_D8, false); // here i am assigning variable with there proper Pin out numbers and setting the output as off
            green = new OutputPort(Pins.GPIO_PIN_D2, false); // here i am assigning variable with there proper Pin out numbers and setting the output as off

            // Initialize the I2C SDA pin
            OutputPort pin = new OutputPort(Pins.GPIO_PIN_SDA, true);
            Thread.Sleep(200);
            pin.Write(true);
            pin.Dispose();

            // Initialize the BMP180 Pressure Sensor
            var sensor = new SensorLibrary.BMP180();
            sensor.Init(SensorLibrary.BMP180.Mode.BMP085_MIDRANGEPOWER);
            rs232_Port = new SerialPort(COM1, 9600, Parity.None); // creating serial Port
            rs232_Port.Open();
            rs232_Port.DataReceived += rs232_Port_DataReceived;
            while (true)
            {
                if (get_Weather)
                {
                    get_Weather = false;
                    // Read pressure and convert to hectoPascals
                    var pressure_hPa = sensor.GetPressure() / 100.0d;

                    // Read temperature in degrees celsius, and convert to fahrenheit
                    var temp_C = sensor.GetTemperature();
                    var temp_F = temp_C * 1.8d + 32;

                    // Read altitude in meters, and convert to feet
                    var altitude_m = SensorLibrary.BMP180.PressureToAltitude(SensorLibrary.BMP180.SENSORS_PRESSURE_SEALEVELHPA, pressure_hPa, temp_C);
                    var altitude_ft = altitude_m * FEET_PER_METER;
                    byte[] outBuffer = Encoding.UTF8.GetBytes("\n\n" + "Barometric Pressure: " + pressure_hPa.ToString("N2") + "\n" + "Temperature: " + temp_F.ToString("N2") + "\n" + "Altitude: " + altitude_ft.ToString("N2") + " f\n\n");
                    rs232_Port.Write(outBuffer, 0, outBuffer.Length);
                    Debug.Print("\n\n" + "Barometric Pressure: " + pressure_hPa.ToString("N2") + "\n" + "Temperature: " + temp_F.ToString("N2") + "\n" + "Altitude: " + altitude_ft.ToString("N2") + " f\n\n");
                }
            }
        }
    }
}
The following snippets of code were provided by Phil Pash, as used in ECET 43400 course.
using System;
using Microsoft.SPOT;

namespace SensorLibrary
{
    public interface ISensor
    {
        bool Init();
    }
}
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace SensorLibrary
{
    public class I2CBus : IDisposable
    {
        private static I2CBus _instance = null;
        private static readonly object LockObject = new object();

        public static I2CBus GetInstance()
        {
            lock (LockObject)
            {
                if (_instance == null)
                {
                    _instance = new I2CBus();
                }
                return _instance;
            }
        }

        private I2CDevice _slaveDevice;

        private I2CBus()
        {
            this._slaveDevice = new I2CDevice(new I2CDevice.Configuration(0, 0));
        }

        public void Dispose()
        {
            this._slaveDevice.Dispose();
        }

        /// <summary>
        /// Generic write operation to I2C slave device.
        /// </summary>
        /// <param name="config">I2C slave device configuration.</param>
        /// <param name="writeBuffer">The array of bytes that will be sent to the device.</param>
        /// <param name="transactionTimeout">The amount of time the system will wait before resuming execution of the transaction.</param>
        public void Write(I2CDevice.Configuration config, byte[] writeBuffer, int transactionTimeout)
        {
            // Set i2c device configuration.
            _slaveDevice.Config = config;

            // create an i2c write transaction to be sent to the device.
            I2CDevice.I2CTransaction[] writeAction = new I2CDevice.I2CTransaction[] { I2CDevice.CreateWriteTransaction(writeBuffer) };

            lock (_slaveDevice)
            {
                int transferred = 0;
                try
                {
                    // the i2c data is sent here to the device.
                    transferred = _slaveDevice.Execute(writeAction, transactionTimeout);
                }
                catch (Exception ex)
                {
                    Debug.Print(ex.Message);
                }

                // make sure the data was sent.
                if (transferred != writeBuffer.Length)
                    throw new Exception("Could not write to device.");
            }
        }

        /// <summary>
    /// Generic read operation from I2C slave device.
    /// </summary>
    /// <param name="config">I2C slave device configuration.</param>
    /// <param name="readBuffer">The array of bytes that will contain the data read from the device.</param>
    /// <param name="transactionTimeout">The amount of time the system will wait before resuming execution of the transaction.</param>
    public void Read(I2CDriver.Configuration config, byte[] readBuffer, int transactionTimeout)
    {
        // Set I2C device configuration.
        _slaveDevice.Config = config;
        // create an i2c read transaction to be sent to the device.
        I2CDriver.I2CTransaction[] readAction = new I2CDriver.I2CTransaction[] { I2CDriver.CreateReadTransaction(readBuffer) };

        lock (_slaveDevice)
        {
            // the i2c data is received here from the device.
            int transferred = _slaveDevice.Execute(readAction, transactionTimeout);
            // make sure the data was received.
            if (transferred != readBuffer.Length)
            { throw new Exception("Could not read from device."); }
        }
    }

    /// <summary>
    /// Read array of bytes at specific register from the I2C slave device.
    /// </summary>
    /// <param name="config">I2C slave device configuration.</param>
    /// <param name="register">The register to read bytes from.</param>
    /// <param name="readBuffer">The array of bytes that will contain the data read from the device.</param>
    /// <param name="transactionTimeout">The amount of time the system will wait before resuming execution of the transaction.</param>
    public void ReadRegister(I2CDriver.Configuration config, byte register, byte[] readBuffer, int transactionTimeout)
    {
        byte[] registerBuffer = { register };
        Write(config, registerBuffer, transactionTimeout);
        Read(config, readBuffer, transactionTimeout);
    }

    /// <summary>
    /// Write array of bytes value to a specific register on the I2C slave device.
    /// </summary>
    /// <param name="config">I2C slave device configuration.</param>
    /// <param name="register">The register to send bytes to.</param>
    /// <param name="writeBuffer">The array of bytes that will be sent to the device.</param>
    /// <param name="transactionTimeout">The amount of time the system will wait before resuming execution of the transaction.</param>
    public void WriteRegister(I2CDriver.Configuration config, byte register, byte[] writeBuffer, int transactionTimeout)
    {
        byte[] registerBuffer = { register };
        Write(config, registerBuffer, transactionTimeout);
        Write(config, writeBuffer, transactionTimeout);
    }

    /// <summary>
    /// Write a byte value to a specific register on the I2C slave device.
    /// </summary>
    /// <param name="config">I2C slave device configuration.</param>
    /// <param name="register">The register to send bytes to.</param>
    /// <param name="value">The byte that will be sent to the device.</param>
    /// <param name="transactionTimeout">The amount of time the system will wait before resuming execution of the transaction.</param>
    public void WriteRegister(I2CDriver.Configuration config, byte register, byte value, int transactionTimeout)
    {
        byte[] writeBuffer = { register, value };
        Write(config, writeBuffer, transactionTimeout);
    }
1 #undef BMP085_USE_DATASHEET_VALS // define for sanity check
2 
3 using System;
4 using Microsoft.SPOT;
5 using Microsoft.SPOT.Hardware;
6 
7 namespace SensorLibrary
8 {
9 
10  public class BMP180 : ISensor
11  {
12   
13   public enum Registers : byte
14   {
15     BMP085_REGISTER_CAL_A1 = 0xAA, // Calibration data (16 bits)
16     BMP085_REGISTER_CAL_A2 = 0xAC, // Calibration data (16 bits)
17     BMP085_REGISTER_CAL_A3 = 0xAE, // Calibration data (16 bits)
18     BMP085_REGISTER_CAL_A4 = 0xB0, // Calibration data (16 bits)
19     BMP085_REGISTER_CAL_A5 = 0xB2, // Calibration data (16 bits)
20     BMP085_REGISTER_CAL_A6 = 0xB4, // Calibration data (16 bits)
21     BMP085_REGISTER_CAL_B1 = 0x86, // Calibration data (16 bits)
22     BMP085_REGISTER_CAL_B2 = 0x88, // Calibration data (16 bits)
23     BMP085_REGISTER_CAL_B3 = 0x8A, // Calibration data (16 bits)
24     BMP085_REGISTER_CAL_MB = 0x8C, // Calibration data (16 bits)
25     BMP085_REGISTER_CAL_MC = 0x8E, // Calibration data (16 bits)
26     BMP085_REGISTER_CAL_MD = 0x90, // Calibration data (16 bits)
27     BMP085_REGISTER_STATUS = 0x01,
28     BMP085_REGISTER_CTRL_REG1 = 0xF4,
29     BMP085_REGISTER_CTRL_REG2 = 0xF5,
30     BMP085_REGISTER_CTRL_REG3 = 0xF6,
31     BMP085_REGISTER_CONFIG = 0xF3,
32     BMP085_REGISTER_PRESS_H1 = 0xF7,
33     BMP085_REGISTER_PRESS_H2 = 0xF8,
34     BMP085_REGISTER_PRESS_H3 = 0xF9,
35     BMP085_REGISTER_PRESS_M1 = 0xFA,
36     BMP085_REGISTER_PRESS_M2 = 0xFB,
37     BMP085_REGISTER_PRESS_S1 = 0xFC,
38     BMP085_REGISTER_PRESS_S2 = 0xFD,
39     BMP085_REGISTER_PRESS_31 = 0xFE
40   }
41 
42   public enum Mode : byte
43   {
44     BMP085_MODE_ULTRALOWPOWER = 0,
45     BMP085_MODE_STANDARD = 1,
46     BMP085_MODE_HIGHERRES = 2,
47     BMP085_MODE_ULTRAHIGHERRES = 3
48   }
49 
50   struct BMP085_Calib_Data
51   {
52     public Int16 ac1;
53     public Int16 ac2;
54     public Int16 ac3;
55     public UInt16 ac4;
56     public UInt16 ac5;
57     public UInt16 ac6;
58     public Int16 b1;
59     public Int16 b2;
60     public Int16 mb;
61     public Int16 mc;
62     public Int16 md;
63   }
64 
65   public const byte BMP085_ADDRESS = 0x77;
66   public const double SENSORS_PRESSURE_SEALEVELHPA = 1013.25;
67   Mode _mode = Mode.BMP085_MODE_ULTRAHIGHERRES;
68   BMP085_Calib_Data _hmp085_Coeffs = new BMP085_Calib_Data();
69 
70   private I2CDevice.Configuration _slaveConfig = new I2CDevice.Configuration(BMP085_ADDRESS, 100);
71   private const int TransactionTimeout = 1000; // ms
72   
73   public bool Init()
74   {
75     return Init(_mode.BMP085_MODE_ULTRAHIGHERRES);
76   }
77 
78   public bool Init(Mode mode)
```c
{
    if ((mode > Mode.BMP085_MODE_ULTRALIGHTRES) || (mode < 0))
    {
        _mode = Mode.BMP085_MODE_ULTRALIGHTRES;
    }
    else
    {
        _mode = mode;
    }
    byte[] whoami = { 0 };
    I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CHIPID, whoami, TransactionTimeout);
    if (whoami[0] != 0x55)
    {
        return false;
    }
    ReadCoefficients();
    return true;
}

void ReadCoefficients()
{
    if (BMP085_USE_DATASHEET_VALS
        _bmp085_coeffs.acl = 468;
        _bmp085_coeffs.ac2 = -72;
        _bmp085_coeffs.ac3 = 14383;
        _bmp085_coeffs.ac4 = 32741;
        _bmp085_coeffs.ac5 = 23153;
        _bmp085_coeffs.b1 = 6198;
        _bmp085_coeffs.b2 = 4;
        _bmp085_coeffs.b3 = -32768;
        _bmp085_coeffs.b4 = 8711;
        _bmp085_coeffs.b5 = 2606;
        _bmp085_mode = 0;
    } else
    {
        byte[] buffer = new byte[2];
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC1, buffer, TransactionTimeout);
        _bmp085_coeffs.acl = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC2, buffer, TransactionTimeout);
        _bmp085_coeffs.ac2 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC3, buffer, TransactionTimeout);
        _bmp085_coeffs.ac3 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC4, buffer, TransactionTimeout);
        _bmp085_coeffs.ac4 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC5, buffer, TransactionTimeout);
        _bmp085_coeffs.ac5 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.AC6, buffer, TransactionTimeout);
        _bmp085_coeffs.ac6 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.B1, buffer, TransactionTimeout);
        _bmp085_coeffs.b1 = (Int16)((buffer[0] << 8) | buffer[1]);
        I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP085_REGISTER_CAL.B2, buffer, TransactionTimeout);
        _bmp085_coeffs.b2 = (Int16)((buffer[0] << 8) | buffer[1]);
    }
```
ctionTimeout);
    _bmp085_coeffs.b2 = (Int16)((buffer[0] << 8) | buffer[1]);
    _bmp085_coeffs.mb = (Int16)((buffer[0] << 8) | buffer[1]);
    _bmp085_coeffs.mc = (Int16)((buffer[0] << 8) | buffer[1]);
    _bmp085_coeffs.md = (Int16)((buffer[0] << 8) | buffer[1]);
#endif
}

UInt16 ReadRawTemperature()
{
    #if BMP085_USE_DATASHEET_VALS
        return 27006;
    #else
        I2CBus.GetInstance().WriteRegister(slaveConfig, (byte)Registers.BMP085_REGISTER_CONTROL, (byte)Registers.BMP085_REGISTER_READTEMP0CMD, TransactionTimeout);
        System.Threading.Thread.Sleep(5);
        byte[] buffer = new byte[2];
        I2CBus.GetInstance().ReadRegister(slaveConfig, (byte)Registers.BMP085_REGISTER_TEMPDATA, buffer, TransactionTimeout);
        return (UInt16)((buffer[0] << 8) | buffer[1]);
    #endif
}

/***************************************************************************/
/*!/ Int32 ReadRawPressure() { */
#endif BMP085_USE_DATASHEET_VALS
    *pressure = 238043;
#endif else
    byte cmd = (byte)((byte)Registers.BMP085_REGISTER_READPRESSURECMD | ((byte)mode << 5));
    I2CBus.GetInstance().WriteRegister(slaveConfig, (byte)Registers.BMP085_REGISTER_CONTROL, (byte)cmd, TransactionTimeout);
    System.Threading.Thread.Sleep(100);
    switch (_mode) {
        case Mode.BMP085_MODE_ULTRALOWPOWER:
            System.Threading.Thread.Sleep(5);
            break;
        case Mode.BMP085_MODE_STANDARD:
            System.Threading.Thread.Sleep(8);
            break;
        case Mode.BMP085_MODE_HIGHER:
            System.Threading.Thread.Sleep(14);
            break;
        case Mode.BMP085_MODE_ULTRAHIGHER:
            default:
                System.Threading.Thread.Sleep(26);
                break;
    }
    byte[] buffer16 = new byte[2];
    I2CBus.GetInstance().ReadRegister(slaveConfig, (byte)Registers.BMP085_REGISTER_PRESSUREDATA, buffer16, TransactionTimeout);
```java
    UInt16 p16 = (UInt16)(buffer15[1] | (buffer15[0] << 8));
    Int32 p32 = (Int32)p16 << 8;
    byte[] buffer8 = new byte[1];
    I2CBus.GetInstance().ReadRegister(_slaveConfig, (byte)Registers.BMP85S_REGISTER_PRESSUREDATA + 2, buffer8, TransactionTimeout);
    byte p8 = buffer8[0];
    p32 |= p8;
    p32 >>= (8 - (byte)_mode);
    return p32;
  }

  public double GetPressure()
  {
    Int32 ut = 0, up = 0, comp = 0;
    Int32 x1, x2, b5, b6, x3, b3, p;
    Uint32 b4, b7;

    /* Get the raw pressure and temperature values */
    ut = ReadRawTemperature();
    up = ReadRawPressure();

    /* Temperature compensation */
    x1 = ((Int32)(ut - (Int32)(_bmp85s_coeffs.ac6)) * ((Int32)(_bmp85s_coeffs.ac5)) / System.Math.Pow(2, 15));
    x2 = ((Int32)(_bmp85s_coeffs.mc * System.Math.Pow(2, 11))) / (x1 + (Int32)(_bmp85s_coeffs.md));
    b5 = x1 + x2;

    /* Pressure compensation */
    b6 = b5 - 4606;
    x1 = (_bmp85s_coeffs.b2 * (b6 * b6) >> 12) >> 21;
    x2 = (_bmp85s_coeffs.ac2 * b6) >> 11;
    x3 = x1 + x2;
    b3 = (((Int32)_bmp85s_coeffs.ac1 * 4 + x3) << (byte)_mode + 2) >> 2;
    x1 = (_bmp85s_coeffs.ac3 * b6) >> 13;
    x2 = (_bmp85s_coeffs.b1 * (b6 * b6) >> 12) >> 16;
    x3 = (x1 + x2 + 2) >> 10;
    b4 = (_bmp85s_coeffs.ac4 * (UInt32)(x3 + 32768)) >> 15;
    b7 = (UInt32)(up - b3) * (50000 >> (byte)_mode);
    if (b7 < 0x80000000)
      p = (Int32)((b7 << 1) / b4);
    else
      p = (Int32)((b7 / b4) << 1);

    x1 = (0 >> 8) * (p >> 8);
    x1 = (x1 * 3830) >> 16;
    x2 = (-7357 * p) >> 16;
    comp = p + ((x1 + x2 + 3791) >> 4);
    return comp;
  }

  public double GetTemperature()
  {
    Int32 X1, X2, B5;  // following ds convention
double t;
    Int32 UT = ReadRawTemperature();

    //if BMP85S_USE_DATASHEET_VALS
    // use data sheet numbers!
    UT = 27898;
    _bmp85s_coeffs.ac5 = 23153;
```
B. Android Software

1. Compiler and Environment

The Android Studio software package is the best choice for developing Android code. It is developed by Google Inc. The language, which is used, is Java. Android Studio also has a GUI (graphical user interface). This is a helpful feature over plain code because it helps the user depict what the application will look like before compiling and downloading to the device. Android Studio is the clear choice when developing Android applications.
2. Code

A simple overview of the Android application is as follows: user will select the LED app on an Android phone. The user will then need to select the Bluetooth module that they are wanting to connect to. For this project, ITEAD is the selected Bluetooth. If it is the first time connecting, user will need to enter a password, which by default is 1234. Once connected, the GUI will show multiple buttons and weather display information. If at any point the user wants to disconnect, they can select the disconnect button and find a new device. Students will begin by using all necessary imports and declaring many variables for the buttons, weather timer, Bluetooth data, and SPP UUID. Immediately in the OnCreate method (provided by Android Studio), the widgets need to be called, along with the timer. The Bluetooth class should be called to connect, and then the buttons’ OnClick methods should be created. Within each OnClick, the corresponding method to turn on the LED should be called. The traffic light timer uses a switch case, like the .Netduino. Within the switch case’s case, the traffic light count should be set to however long the user wants the corresponding LED to be on. To get the weather, a method is called and a ‘t’ is sent to the .Netduino application, telling it to send the weather back to the Android phone.
package com.rlibby.etic.rlibby.led;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;
import android.os.Handler;

import java.io.InputStream;
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;

public class ledControl extends AppCompatActivity {

    // Buttons
    ImageButton Disconnect, Green, Off, Red, Trafficlight, Yellow;
    Button Weather;

    // Traffic light timer
    Timer myTimer = new Timer();
    private boolean isTrafficlight = false;
    private int trafficlightCnt = 0;
    private int whichLight = 0;

    // Weather timer
    private int weatherUpdate = 1;
    TextView PressureText, TempText, AltitudeText;

    // Bluetooth data
    String address = null;
    private ProgressDialog progress;
    private BluetoothAdapter myBluetooth = null;
    private BluetoothSocket btSocket = null;
    private boolean btConnected = false;
    byte[] btBuffer = new byte[1024];
    Handler bluetoothIn;
    final int handlerState = 0;
    private StringBuilder recDataString = new StringBuilder();

    private ConnectedThread mConnectedThread;

    // SPP UUID. Look for it
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent newInt = getIntent();
        address = newInt.getStringExtra(DeviceList.EXTRA_ADDRESS); // receive the address of the bluetooth device

        // view of the ledControl
        setContentView(R.layout.activity_led_control);

        // call the widgets
        // About = [ImageButton]findViewById(R.id.led_about);
        // Disconnect = [ImageButton]findViewById(R.id.disconnect);
        // Green = [ImageButton] findViewById(R.id.green);
    }
Off = (ImageButton) findViewById(R.id.off);
Red = (ImageButton) findViewById(R.id.red);
Trafficlight = (ImageButton) findViewById(R.id.trafficlight);
Yellow = (ImageButton) findViewById(R.id.yellow);
Weather = (Button) findViewById(R.id.weatherButton);
PressureText = (TextView) findViewById(R.id.pressureTextView);
TempText = (TextView) findViewById(R.id.tempTextView);
AltitudeText = (TextView) findViewById(R.id.altitudeTextView);
myTimer.scheduleAtFixedRate(new myTimerTask(), 0, 1000);
new ConnectBT().execute(); //Call the class to connect
//commands to be sent to bluetooth

Red.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        isTrafficlight=false;
        turnOnRedLed(); //method to turn on
    }
});

Green.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        isTrafficlight=false;
        turnOnGreenLed();
    }
});

Yellow.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        isTrafficlight=false;
        turnOnYellowLed();
    }
});

Off.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        isTrafficlight=false;
        turnOffLed(); //method to turn off
    }
});

Disconnect.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Disconnect(); //close connection
    }
});

Weather.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        WeatherUpdate(); //close connection
    }
});

/*About.setOnClickListener(new View.OnClickListener() {*/
    @Override
}
public void onClick(View v) {
    abt(v);
}

Trafficlight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        trafficlightctrl(v);
    }
});

bluetoothIn = new Handler()
{
    public void handleMessage(android.os.Message msg)
    {
        String readMessage = (String) msg.obj; // msg.arg1 = bytes from c
        recDataString.append(readMessage); // keep appending to string
        int endOfLineIndex = recDataString.indexOf("-"); // determine the end-of-line
        if (endOfLineIndex > 0) // make sure there data before -
        {
            //TempText.setText(recDataString);
            if (recDataString.charAt(0) == 'F')
            
            "// if it starts with F we know it is what we are looking for/
            
            Double temp = Double.parseDouble(recDataString.substring(1, 6));
            // get sensor value from string between indices 1-5
            Double pressure = Double.parseDouble
            (recDataString.substring(7, 13));
            // same again...
            Double altitude = Double.parseDouble
            (recDataString.substring(14, 28));
            TextTemp.setText("Temperature = " + temp.toString() + "F");
            // update the textviews with sensor values
            PressureText.setText("Pressure = " + pressure.toString() + "mPa");
            AltitudeText.setText("Altitude = " + altitude.toString() + "feet");
            recDataString.delete(0, recDataString.length()); // clear all string data
        }
    }
}

https://wingsooharry.wordpress.com/2014/04/15/android-sendreceive-data-with-arduino-using-bluetooth-part-

private void trafficlightctrl(View v) {
    if(!isTrafficlight) // traffic light enabled
    {
        turnOffLed();
        trafficlight_cnt = 4; // 4 seconds to zero
        whichLight = 1; // goes to case 2 in switch case which is red
        turnOnRedLed(); // turns on red led
        isTrafficlight=true;
    }
}

private class myTimerTask extends TimerTask
{
    public void run()
{  
  if (isTrafficlight) // only execute if active
  
  
  {  
    trafficlight cnt--; // decrement count
    if (trafficlight_cnt == 0)
    
    {  
      switch (whichLight) // switch on whichLight
      
      {  
        default:
        
        case 0: // green is on
         trafficlight cnt = 2; // on for 2 seconds
          whichLight = 1; // go to case 1 (yellow) next
          turnOnYellowLed(); // turn on yellow led
          break;
        
        case 1: // yellow is on
         trafficlight cnt = 4; // on for 4 seconds
          whichLight = 3; // go to case 2 (red) next
          turnOnRedLed(); // turn on red led
          break;
        
        case 2: // red is on
         trafficlight cnt = 5; // on for 5 seconds
          whichLight = 0; // go to case 0 (green) next
          turnOnGreenLed(); // turn on green led
          break;

        }  
      
      }  
    
    if (---weather update == 0) // update weather every 10 seconds
    
    {  
      // time to update weather
      weather update = 10; // every 10 seconds
      WeatherUpdate(); // call get weather function
    }  
  }  

  private void WeatherUpdate()
  
  {  
    if (btSocket != null)
    
    {  
      try{
        btSocket.getOutputStream().write("t", getBytes()); // send 't' to netduino to retrieve weather data
        
    catch (IOException e)
      
      {  
        msg("Error");
      }  
    }  
  }  

  /*public void abt(View v)
  
  {  
    if(v.getId() == R.id.about)
    
    {  
      Intent i = new Intent(this, AboutActivity.class);
      startActivity(i);
    }  
  }  
*/

  private void Disconnect()
  
  {  
    if (btSocket != null) // If the btSocket is busy
    
    {  
      try
      
      {  
        btSocket.close(); // close connection
      }  
    
    catch (IOException e)
      
    {  
      msg("Error");
    }  
  }
    finish(); //return to the first layout

    private void turnOffLed()
    {
        if(btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("0").getBytes);
            } catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void turnOnRedLed()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("R").getBytes);
            } catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void turnOnYellowLed()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("Y").getBytes);
            } catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void turnOnGreenLed()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("G").getBytes);
            } catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    // fast way to call Toast
    private void msg(String s)
    {
        Toast.makeText(getApplicationContext(), s, Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.menu_led control, menu);
return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
// Handle action bar item clicks here in the appropriate place of the activity.
int id = item.getItemId();

//noinspection SimplifiableIfStatement
if (id == R.id.action_settings) {
    return true;
}

return super.onOptionsItemSelected(item);
}

public class ConnectBT extends AsyncTask<Void, Void, Void> // UI thread
{
    private boolean ConnectSuccess = true; // if it's here, it's almost connected

    @Override
    protected void onPreExecute()
    {
        progress = ProgressDialog.show(ledControl.this, "Connecting...", "Please wait!!!"); // show a progress dialog
    }

    @Override
    protected Void doInBackground(Void... devices) // while the progress dialog is shown, the connection is done in background
    {
        try
        {
            if (btSocket == null || !isBtConnected)
            {
                myBluetooth = BluetoothAdapter.getDefaultAdapter(); // get the mobile bluetooth device
                BluetoothDevice dispositivo = myBluetooth.getRemoteDevice(address); // connects to the device's address
                btSocket = dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID); // create a RFCOMM (SPP) connection
                BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                btSocket.connect(); // start connection
                mConnectedThread = new ConnectedThread(btSocket);
                mConnectedThread.start();
            }
        }
        catch (IOException e)
        {
            ConnectSuccess = false; // if the try failed, you can check the exception here
            return null;
        }
    }

    @Override
    protected void onPostExecute(Void result) // after the doInBackground, it checks if everything went fine
    {
        super.onPostExecute(result);

        if (!ConnectSuccess)
        {
            msg("Connection Failed. Is it a SPP Bluetooth? Try again.");
            finish();
        }
        else
        {
            msg("Connected.");
            isBtConnected = true;
        }
    }
IV. TESTING

The best method of testing in the practice of coding is to write small bits of code and ensure they work on their own. For example, the student wrote the BLANK METHOD and confirmed that it worked separate of all other code. This helps ensure that there are no problems with each piece of code and greatly reduces the amount of time needed to debug the entire system.
spent troubleshooting errors when all the code comes into one program. When errors do occur in the final program the coder will know that each piece of code works on its own and that the errors are likely being caused by issues with the pieces working together instead of the code inside of any given piece.

III. CONCLUSIONS AND RECOMMENDATIONS

At the conclusion of this project, the senior student will have developed lecture materials, lab procedures, and working applications of both Android and .Netduino. The senior student will have learned how a Bluetooth shield works and its communication protocol, have a better understanding of coding in C# using Visual Studio 2015 in the .Net Micro Framework environment and Android in the Android SDK, and how to communicate between devices in different languages on different hardware.
LIST OF REFERENCES


